Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра захисту інформації

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему: «Модель і засіб виявлення вразливостей у Web-додатках»

08-20.МКР.016.00.000.ПЗ

Виконав: студент групи 1БС-18м

Спеціальність 125 – Кібербезпека

ОПП Безпека інформаційних і комунікаційних систем

_____Юсуфов Р.Ю.

Керівник: к. т. н., доц. каф. ЗІ

_____ Войтович О.П.

Рецензент к. т. н., проф. доц. кафедри ОТ

_____ Азарова А.О.

Вінниця – 2019 року

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень підготовки – магістерський
Спеціальність – 125 Кібербезпека
Освітня програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ, д. т. н., проф.
_____ В. А. Лужецький
_____ 2019 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Юсуфову Руслану Юрійовичу

1. Тема роботи: «Модель і засіб виявлення вразливостей у Web-додатках» керівник роботи: Войтович Олеся Петрівна, к. т. н., доц. каф. ЗІ, затверджена наказом ВНТУ №254 від 02.10.2019 року
2. Строк подання студентом роботи _____ 2019 р.
3. Вихідні дані до роботи:
– модель і засіб виявлення вразливостей у Web-додатках;
– операційна система – Windows;
– версія для використання за персональним комп'ютером;
– мова програмування JavaScript;
– середовище розробки – Visual Studio Code
4. Зміст розрахунково-пояснювальної: Вступ. Науково-дослідне обгрунтування. Розробка моделей виявлення загроз. Розробка та тестування програмного засобу. Економічна частина. Висновки. Перелік використаних джерел. Додатки.
5. Перелік Іллюстративного матеріалу.
Модель виявлення вразливостей до XSS-атак (плакат, А4). Модель виявлення вразливостей до SQL-ін'єкцій (плакат А4). Модель виявлення крітоджекера (плакат, А4). Алгоритм порівняння значень із пороговим (плакат А4). Алгоритм порівняння значень множин (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Войтович О. П., к. т. н., доц. каф. ЗІ		
2	Войтович О. П., к. т. н., доц. каф. ЗІ		
3	Войтович О. П., к. т. н., доц. каф. ЗІ		
4	Мацкевічус С.С., ст. викл. Каф. ЕПВМ		

7. Дата видачі завдання _____ 2019 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів Магістерської кваліфікаційної дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2019 – 04.09.2019	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	05.09.2019 – 15.09.2019	
3	Науково-технічне обґрунтування	16.09.2019 – 22.09.2019	
4	Розробка технічного завдання	23.09.2019 – 29.09.2019	
5	Розробка рішень	30.09.2019 – 12.10.2019	
6	Практична реалізація, моделювання, експериментування, результати	14.10.2019 – 10.11.2019	
7	Розробка розділу економічного обґрунтування доцільності розробки	11.11.2019 – 17.11.2019	
8	Аналіз виконання ТЗ, висновки	18.11.2019 – 24.11.2019	
9	Оформлення пояснювальної записки	25.11.2019 – 30.11.2019	
10	Попередній захист та доопрацювання МКР	28.11.2019 – 01.12.2019	
11	Перевірка магістерської роботи на наявність плагіату	02.12.2019 – 10.12.2019	
11	Представлення МКР до захисту	02.12.2019 – 10.12.2019	
11	Захист МКР	02.12.2019 – 10.12.2019	

Студент _____ Юсуфов Р.Ю.

(підпис)

Керівник роботи _____ Войтович О. П.

(підпис)

АНОТАЦІЯ

УДК 004.056

У магістерській кваліфікаційній роботі досліджені найпопулярніші вразливості сучасних веб-ресурсів, їх вплив шкідливий вплив на роботу сайтів і кінцевого користувача, а також були проаналізовані вже існуючі методи їх визначення. Робота спрямована на покращення кібербезпеки у веб-просторі шляхом виявлення небезпечних або потенційно небезпечних веб-ресурсів. У ході магістерської кваліфікаційної роботи було розроблено модель програмний засіб, що реалізує пошук небезпечних або потенційно небезпечних веб-ресурсів шляхом їх сканування на найпоширеніші атаки. Програмний засіб розроблений у середовищі Visual Studio Code мовою JavaScript.

ABSTRACT

The master's qualification work investigates the most popular vulnerabilities of modern web resources, their impact, the harmful impact on the work of sites and the end user, as well as the existing methods of their determination. The work is aimed at improving cybersecurity in the web by identifying dangerous or potentially dangerous web resources. In the course of the master's qualification work, a model software was developed that realizes the search for dangerous or potentially dangerous web resources by scanning them for the most common attacks. The software is developed in the Visual Studio Code environment in JavaScript.

ЗМІСТ

ВСТУП

Поширення інтернету сприяє стрімкому розвитку Web-додатків, які охоплють різні аспекти життя: відеохостінги, форуми, соціальні мережі, інтернет-банкінг та багато іншого. Зручність доступу до будь-яких ресурсів приваблює велику кількість населення, але також сприяє розвитку різноманітних видів атак на ці ресурси. Особисті дані у соціальних мережах, рахунки та інша персональна або навіть конфіденційна інформація приваблює хакерів, які намагаються заволодіти цими ресурсами заради своєї вигоди.

Існує велика кількість вразливостей Web-додатків, найпоширенішими з них є:

– XSS-атаки. XSS працює в браузері користувача і дає можливість вкрасти його інформацію. XSS або Cross-Site Scripting працює в JavaScript за тим же принципом що і ін'єкції. Хакер передає спеціальну рядок в будь-якому полі, в рядку JS код, браузер вирішує, що цей код відправлений сайтом і запускає його.

– SQL-ін’єкції – це вразливості, які з'являються в процесі передачі неперевірених, введених користувачем даних інтерпретатора для виконання. Тобто будь-який користувач може виконати довільний код у інтерпретаторі. За допомогою них шахрай отримує доступ до бази даних, читає приховані дані та може навіть записувати свої значення.

– Кріптоджекінг – це онлайн-загроза, що з'явилася відносно недавно. Пов'язані з цією загрозою шкідливі об'єкти ховаються на комп'ютерах або мобільних пристроях і використовують їх ресурси для видобутку («Майнінг») електронних грошей – криптовалюта. Ця загроза активно розвивається,

набуваючи нових форм: захоплює інтернет-браузери, вражає будь-які типи пристроїв - від настільних комп'ютерів і ноутбуків до смартфонів і навіть мережевих серверів.

Як і у випадку з іншими атаками шкідливого ПО, головним мотивом зловмисника є прибуток, але на відміну від інших загроз, шкідливі об'єкти цього типу намагаються приховати свою присутність від користувача [1].

Головною метою дипломної роботи полягає розробка ефективної протидії переліченим видам інтернет-загроз, яка буде реалізована у якості програми для аналізу веб-ресурсів. Основними завданнями програмного засобу є:

– моніторинг веб-сторінок браузеру на наявність даних загроз;

– виявлення загроз та попередження користувача про їх наявність;

– аналіз загрози та надання користувачеві вибору пропозиції щодо вирішення проблеми.

1 НАУКОВО-ДОСЛІДНЕ ОБГРУНТУВАННЯ

1.1 Актуальність теми

Через зріст популярності веб-ресурсів у сучасний час з’являється і велика кількість атак, які використовують їх вразливості.

Статистика ІБ-фахівці з White Hat Security свідчить про те, що, власники сайтів часто не думають про захист своїх ресурсів, незважаючи на те, що вони зазнають збитків від зламів сайтів та витік даних клієнтів. Найпоширенішою вразливістю виявилася XSS (Cross-Site Scripting) – її виявлено на 70% сайтів. Друга за поширеністю вразливість пов'язана з витоком службових даних - 45% сайтів «викриті» в її наявності. Під цією вразливістю White Hat Security має на увазі доступ до інформації службового характеру: інформація користувачів, вихідні коди скриптів, дані про серверному ПО, а також про помилки в ньому. На 25% сайтів виявлені вразливості, дозволяють спамерам експлуатувати веб-ресурси в своїх цілях. Найчастіше ця проблема стосується блогів і чатів [1].

SQL injection – вразливість, що виникає як наслідок недостатньої перевірки прийнятих від користувача значень, в скрипті або програмі. Впровадження SQL, в залежності від типу використовуваної СУБД і умов впровадження, може дати можливість атакуючому виконати довільний запит до бази даних (наприклад, прочитати вміст будь-яких таблиць, видалити, змінити або додати дані), отримати можливість читання і / або запису локальних файлів і виконання довільних команд на атакується сервері. Атака типу впровадження SQL може бути можлива через некоректні обробки вхідних даних, що використовуються в SQL-запитах.

Розробник прикладних програм, що працюють з базами даних, повинен знати про такі вразливості і вживати заходів протидії запровадженню SQL[2].

Атаки цього типу несуть основну загрозу для сервера і бази даних, але опосередковано може також вплинути на цілісність особистих даних, тому що зловмисник може отримати доступ до всієї бази. Це становить загрозу для

звичайного користувача, адже не захищена база даних може спричинити витік особистої інформації або навіть конфіденційної [3].

Кріптоджекінг – відносно нове явище, яке є однією з найпоширеніших онлайн-загроз. В блозі Malwarebytes опублікована доповідь науково-дослідного відділу, згідно з яким з вересня 2017 року шкідливий Майнінг криптовалют (ще одне позначення кріптоджекінга) знаходиться на першому місці в рейтингу найбільш часто виявляються шкідливих об'єктів. В жовтні 2017 року журнал Fortune прогнозує збільшення популярності кріптоджекінг і його становлення у якості найсерйознішої загрози в онлайн-світі. В першому кварталі 2018 роки кількість виявлених шкідливих програм для кріптоджекінга на платформі Android зросла на 4000 відсотків[3].

Низький поріг входу на ринок чорного заробітку на нелегальному Майнінгу призводить до того, що здобиччю криптовалюти займаються люди без технічних знань і досвіду участі в шахрайських схемах. Більшість тих, хто займається кріптоджекінгом, не вважають це злочином. Законодавство має достатню кількість лазівок для уникання переслідування цей вид розкрадання. Більшість методів встановлення програми-Майнера припускають порушення законів, але випадків арешту і практики судового переслідування за кріптоджекінг одиниці.

Дані вразливості є дуже популярними, особливо сильно набирає популярність кріптоджекінг через швидке поширення кріптовалют. Нижче будуть детально розглянуті ці вразливості.

1.2 SQL-ін’єкції

SQL -injection являє собою посил будь-яких довільних запитів до бази даних. На читання, на запис і так далі. Тобто в основному для перегляду і пошуку важливих записів в БД, а також їх розповсюдженням з корисною метою.

Ін’єкції розподіляються на такі види:

– Класична SQL: хакер розкриває базу даних, що дозволяє передивлятись будь-яку інформацію.

– Заснована на виведення помилок (error-based SQL): більш складний у реалізації варіант. Заснований він на отриманні іниормації про базу, її дані за допомогою виведення помилок СУБД. Але працює тільки у випадку, якщо при конфігурації не був відключений висновок помилок.

– Заснована на логіці запитів (bool ean- based SQL): або "сліпа ін'єкція". В параметр додається підзапит, на який БД буде відповідати або True, або False. Атака заснована на переборі, за допомогою якого можна витягнути назву таблиць, колонок з БД. Недоліком є зростання часу перебору із зростанням атакованої бази.

– Заснована на часі (time-based SQLi): також є різновидом сліпої ін'єкції, але має удосконаленості. Додається підзапит, що приводить до уповільнення або паузі роботи БД. Далі порівнюється час відповіді на True і на False запити, і так символ за символом можна отримати весь вміст БД, але часу піде на це ще більше, ніж в разі boolean-based атаки.

– Out-of-band SQL i: рідкісний тип вразливості. Атака може бути успішна тільки при певних обставинах, якщо сервер БД може генерувати DNS або HTTP запити. Експлуатується вразливість теж перебором.

Такого роду атаки проводяться в першу чергу для «зливу» БД, коли базу даних продають певним рекламним компаніям або іншим зацікавленим особам. Це найчастіше і популярне явище. Це може бути серйозною загрозою, якщо це база з великою кількістю акаунтів, в яких вказані адреси, телефони, номери карток із супутньою інформацією тощо. Подібні атаки завдають великі удари по репутації і фінансам сервісів.

1.3 XSS-атаки

Це такий тип атак, який впроваджує в веб-системи шкідливий код, змушуючи її видавати змінені дані, підміняє посилання (видимі / приховані) або виводить власну рекламу на ураженому ресурсі.

Існує два напрямки атак:

– Пасивні – атаки, які вимагають безпосереднього втручання суб'єкта атаки. Суть полягає в тому, щоб змусити жертву перейти по шкідливої

посиланням для виконання шкідливого коду. Такий тип атак складніший в реалізації, адже необхідно володіти не тільки технічними, але і психологічними знаннями.

– Активні – це вид атак, коли хакер намагається знайти вразливість в фільтрі сайту. Потрібно за допомогою комбінації тегів і символів створити такий запит, щоб сайт його зрозумів і виконав команду. Після винайдення вразливості захисту, в запит можна вкласти шкідливий код, який, може красти cookie і пересилати туди, куди вкаже зломисник.

Чіткої класифікації для міжсайтового скриптинга не існує, але виділено три основних типи:

– Збережені XSS (постійні). Один з найнебезпечніших типів вразливостей, так як дозволяє зловмисникові отримати доступ до сервера і вже з нього управляти шкідливим кодом (видаляти, модифікувати). Кожен раз при зверненні до сайту виконується заздалегідь завантажений код, який працює в автоматичному режимі. В основному таким вразливостям схильні форуми, портали, блоги, де присутня можливість коментування в HTML без обмежень. Шкідливі скрипти з легкістю можуть бути вбудовані як в текст, так і в картинки, малюнки.

– Відображені XSS (непостійні). В цьому випадку шкідлива рядок виступає в ролі запиту жертви до зараженого веб-сайту. Працює цей принцип за наступною схемою:

a) Зловмисник заздалегідь створює URL-посилання, яка буде містити шкідливий код і відправляє його своїй жертві.

b) Вона спрямовує цей URL-запит на сайт (переходить за посиланням).

c) Сайт автоматично бере дані з шкідливою рядки і підставляє у вигляді модифікованого URL-відповіді жертві.

d) У підсумку в браузері у жертви виконується шкідливий скрипт, який і міститься у відповіді, а зловмисник отримує все cookies цього користувача.

– DOM-моделі. У цьому варіанті можливе використання як збережених XSS, так і відображених. Суть полягає в наступному:

a) Зловмисник створює URL-адресу, який заздалегідь містить шкідливий код, і відправляє його по електронній пошті або будь-яким іншим способом користувачеві.

b) Людина переходить за цим посиланням, заражений сайт приймає запит, виключаючи шкідливу рядок.

c) На сторінці у користувача виконується сценарій, в результаті чого завантажується шкідливий скрипт і зловмисник отримує cookies[4].

Для швидкої перевірки сайту на наявність вразливостей XSS можна скористатися спеціалізованими сервісами, які в автоматичному режимі проведуть сканування сторінки. В обов'язковому порядку потрібно перевіряти всі URL, де можлива відправка даних з боку користувача (форми коментарів, зворотний зв'язок, пошук).

Подібні сервіси не дають повної гарантії успіху, тому рекомендується перевіряти знайдені сторінки в ручному режимі і виключити всі небезпечні спеціальні символи, замінивши їх безпечними. Йдеться про дужки для тегів – <i>, в яких прописуються всі зарезервовані мовою html-запити і теги[4].

1.4 Кріптоджекінг

1.4.1 Кріптовалюти

Складений з двох слів – «криптографія» і «валюта» - термін «криптовалюта» позначає електронні гроші, застосування яких ґрунтується на принципах комплексного математичного шифрування. Вся криптовалюта існує у вигляді зашифрованих децентралізованих грошових одиниці, які можуть вільно передаватися між учасниками мережі. Криптовалюта – це електроенергія, перетворене в рядки коду, які володіють реальною грошовою вартістю.

Одиниці криптовалюта називаються монетами та представляють собою записи в базі даних. Щоб здійснити транзакцію і внести зміни в базу даних, потрібно виконати ряд умов. Коли користувач авторизує переказ коштів, знімаєте гроші з рахунку або депозиту, банківська база даних оновлюється за

рахунок інформації про нові транзакціях. Криптовалюта працює аналогічним чином, однак їх база даних децентралізована.

На відміну від традиційних валют, криптовалюта не забезпечена державою або банком. До них не застосовується державний нагляд або політика центрального регулятора. Криптовалюти децентралізовані: вони контролюються безліччю копій однієї бази даних, які одночасно працюють в мережі, що нараховує мільйони комп'ютерів. Тому така база даних не належить жодній особі або організації. База даних криптовалюти функціонує як цифровий бухгалтерський журнал. Вона використовує шифрування, щоб контролювати створення нових монет і перевіряти перекази коштів. При цьому забезпечується постійна і повна анонімність операцій з криптовалютою та її власників[1].

Власники криптовалюти зберігають свої гроші в віртуальних «гаманцях», зашифрованих за допомогою персональних ключів. Щоб переказати кошти між власниками двох електронних гаманців, запис про цю транзакції повинна бути внесена в децентралізований цифровий журнал. Спеціальні комп'ютери у певний проміжок часу, який становить близько десяти хвилин, збирають дані про останні операції з біткоїнами і іншими криптовалютами і перетворюють їх в зашифрований вигляд. Потім ця зашифрована транзакція очікує підтвердження.

Переказ коштів підтверджується тільки в тому випадку, якщо хто-небудь з членів-майнерів іншої категорії користувачів незалежно отримає рішення цього завдання, яке підтверджує легітимність транзакції, після чого кошти переходять з одного гаманця на інший. При цьому кожна особа одночасно працюючих майнерів прагне розшифрувати більшу кількість інформації, щоб підтвердити транзакцію[2].

1.4.2 Майнінг та кріптоджекінг

Майнінг, також видобування — діяльність з підтримки розподіленої платформи і створення нових блоків з можливістю отримати винагороду у формі емітованої валюти и комісійних зборів у різних криптовалютах, зокрема в Біткоїнах. Вироблені обчислення потрібні для забезпечення захисту від

повторного використання одних і тих же одиниць валюти, а зв'язок майнінгу з емісією стимулює людей витрачати свої обчислювальні потужності і підтримувати роботу мереж[5].

Є різні типи майнінгу:

– GPU-Майнінг, тобто Майнінг на відкритих;

– CPU-Майнінг, тобто Майнінг на процесорі;

– Майнінг на АСІКах (ASIC) – спеціальному обладнанні, створеному для видобутку криптовалюта, що працюють на певних алгоритмах.

Кріптоджекінг – це схема використання чужих пристроїв (комп'ютерів, смартфонів, планшетних ПК або навіть серверів) без відома їх власників з метою прихованого майнінгу криптовалюти. Замість того щоб будувати спеціалізовані комп'ютерні системи для видобутку криптовалюти, хакери вдаються до методів кріптоджекінга і викрадають обчислювальні потужності з пристроїв своїх жертв. Складаючи всі ці потужності, хакери можуть успішно (а головне - без істотних витрат) конкурувати з великими гравцями на ринку видобутку криптовалюти.

Користувач може не помітити як став жертвою кріптоджекінга. У більшості випадків програми для кріптоджекінга прагнуть приховати свою активність від користувача, однак це не робить їх нешкідливими. Крадіжка обчислювальної потужності уповільнює роботу комп'ютера, підвищує рахунки за електроенергію і скорочує термін експлуатації пристрою. У більшості випадків користувач може помітити певні ознаки вторгнення, якщо його пристрій починає працювати повільно або використовувати вентилятор охолодження частіше звичайного, виною цьому може бути кріптоджекінг.

Майнінг криптовалюти є дуже прибутковим, проте сьогодні навіть мати зиск вкрай важко в силу величезних супутніх витрат. Злочинці в обхід перелічених труднощів залучаються до кріптоджекінгу, який стає ефективним і недорогим способом отримати кріптомонети.

Кріптоджекери (програми для кріптоджекінга) використовують кілька способів заразити комп'ютер. Один з них схожий на методи класичного шкідливого ПЗ. Користувач переходить за шахрайською посиланням в

електронному листі - і код для майнінгу завантажується безпосередньо на комп'ютер. Коли комп'ютер заражений, кріптоджекер запускається і починає безперервно добувати заповітну криптовалюту, залишаючись загубленим серед фонових процесів. Локальна робота об'єктів даного типу дозволяє зарахувати їх до постійних погроз, які представляють небезпеку для комп'ютера.

Альтернативний спосіб кріптоджекінга іноді називають тіньовим Майнінгом. Як і в разі шкідливої реклами, що використовує уразливість системи, дана схема передбачає впровадження невеликого коду JavaScript у веб-сторінку. Це дозволяє запустити видобуток криптовалюта на комп'ютері користувача, який відвідав відповідну веб-сторінку.

Тіньовий Майнінг криптовалюти може вразить навіть мобільні пристрої Android. В цьому випадку Зловмисники вдаються до тих же методів, що і при атаці на настільні комп'ютери. Це може бути Троянська програма, прихована в завантаженому додатку. Або користувач телефону може бути перенаправлений на заражені веб-сайт, який залишить в системі таке ж непомітною спливаюче вікно. Крім того, існує троянська програма, яка проникає в систему Android і приносить з собою вкрай агресивний установник; він в свою чергу перевантажує процесор, викликає надмірні нагрівання телефону і виснажує акумулятор (іноді навіть викликаючи його деформації). Такі дії сприяють швидкому псуванню пристрою, що робить його непридатним.

Більшість мобільних пристроїв має невелику потужність, тому майнити за допомогою цих гаджетів невигідно. Це вірно для окремо взятого пристрою. Але якщо атаки носять масовий характер, то велика кількість смартфонів генерує саме значний обсяг ресурсів.

Також потрапляють під загрозу пристрої розумного будинку[9]. Такі прилади як розумний холодильник, телевізор, лампа, розетка починають оснащуватися процесорами і мають з’єднання з мережею інтернет, що робить їх також потенціальними цілями для атак. Поодинці такі гаджети не зможуть принести користі, але у масовому використанні вони зможуть надати велику потужність.

1.4.3 Аналіз проблеми кріптоджекінгу

Все, що робить кріптоджекінг – видобуває криптовалюту приховано від користувача, на пристрої якого цей процес відбувається. Цей вид атаки не намагається вкрасти особисті дані або зламати профілі у соціальних мережах, поштових скриньках та не несе ніякої прямої загрози. Головна неприємність, яку створює майнер майже одразу, це уповільнення роботи пристрою, тому що обчислення для видобування криптовалюти потребують значних потужностей.

Використання великих потужностей впливає на доступність інформації, тому що вся робота мікпропроцессора витрачається на обчислення хешів.

Інші побічні ефекти з'являються згодом, такі як: підвищені тарифи за електрику, швидкий знос та перехід до повної непридатності відеокарти та процесора комп'ютера.

Кристал, який є основою напівпровідникових чіпів, не містить механічних елементів, схильних до природного зносу, тому постійна робота не пошкоджує відеоркарти і процесори. Основною проблему чіпів є несприятливі умови роботи. Головними чинниками є температурний режим і напруга. Якщо вони сильно відхиляються від норми, транзистори в складі кристала деградують, приводячи до його виходу з ладу. Крім самого кристала, перегрів також здатний пошкодити пайку чіпа[6].

Високі температури (близько 90-100 градусів), а також їх різкий перепад, призводять до того, що в кульках припою, якими кріпиться чіп, виникають мікротріщини і оксиди, міцність пайки погіршується. В результаті фіналом цього процесу стає так званий «відвал чіпа» - втрата частиною його висновків контакту з платою. Це – головна причина, по якій відеокарти виходять з ладу.

Піддаються зносу і елементи живлення відеокарти (резистори, транзистори, конденсатори, діоди і т. Д.). Якщо напруги живлення недостатньо, а в силових лініях виникають просадки по вольтажу, то сила струму збільшується, що призводить до зростання теплових втрат на опорі. В результаті живлять елементи перегріваються, з часом деградуючи і виходячи з ладу[6, 7].

«Білі» майнери працюють на спеціальних «фермах» для Майнінгу, встановлених в сухому прохолодному приміщеннях, де комп'ютери майже не піддаються перепадам температур. Також встановлюється швидкість вентиляторів на 100%, в результаті чого GPU працюють стабільно при температурі близько 70 ° С. Та й сама ферма зазвичай має відкриту конструкцію, що сприяє кращому охолодженню її комплектуючих. Всі ці умови сприяють довгій роботі апаратних частин, але вони дотримуються людьми, які свідомо йдуть на такі міри, людина, яка нічого не підозрює, що її комп'ютер використовується зловмисниками, не буде проводити подібних маніпуляцій. Саме це сприяє швидкому виходу з ладу пристрою.

Смартфони також підпадають під ризик бути використаними зловмисниками. У цьому випадку дуже швидко деградує акумулятор. Через гірший тепловідвід у ситуаціях, коли пристрій знаходиться у кишені, або накритий речами, а також велике навантаження на процесор, створює ймовірність займання або навіть вибуху батарею. Навіть у разі звичайної деградації акумулятора, сучасні смартфони мають закритий корпус, що ускладнює заміну частин і коштує дорожче.

Прихований майнінг обмежує легітимного користувача низькою швидкістю роботи пристрою, заважає адекватно взаємодіяти людину зі комп'ютером, смартфоном або іншим гаджетом, що має з'єднання до інтернету. Постійне перебування під керуванням прихованого майнера призводить і до матеріальних втрат на ремонт або покупку нового пристрою.

Це також загроза стійкості і безперервності бізнес-процесів в силу уповільнення роботи корпоративних систем і підвищеної амортизації апаратних засобів. Зараження інфраструктури трояном-Майнером може привести до відмови корпоративних додатків, мереж і систем. Несанкціонована робота сторонніх програм без відома власників бізнесу загрожує репутаційним втратам, а також ризиками з боку комплаенса і регуляторів.

1.5 Існуючі шляхи виявлення та вирішення проблеми

1.5.1 Шляхи виявлення

Виявити кріптоджекінг не дуже складно, так як розробники ПЗ для цього заняття не можуть зробити його повністю потайним. Перший непрямий привід запідозрити, що на комп'ютері хтось використовує потужності для майнінгу – зниження загальної продуктивності.

Перший аналіз графіку завантаження процесорних ядер у диспетчері задач може вказати на проблему, якщо комп'ютер став працювати помітно гірше, програми та сайти відкриваються довше, у випадку, якщо на даний момент не працює ніякої важкий софт (компілятори, декодери аудіо і відео і т. Д.), А відкритий тільки браузер – середнє завантаження ЦП не повинне перевищувати 10-25%, причому на всі ядра рівномірно.

Якщо навантаження на ядра в фоновому режимі помітно вище зазначеної, або ж частина з них задіяні на 100% - це ознака, що на ПК можуть майнити. Аналіз розділу процесів може показати, який з них навантажує ядра. Якщо цей процес має невідоме походження, або це один з процесів браузера, то велика ймовірність, що ПК піддається кріптоджекінгу.

На смартфоні виявити кріптоджекінг можна за допомогою аномальної активності окремих програм. Якщо апарат став працювати повільніше, а розряджатися швидше, можна проаналізувати споживання енергії додатками. Якщо якась програма споживає дуже багато – велика ймовірність, що в неї вбудований код для прихованого Майнінгу. Але такі підходи не гарантують, що ПЗ буде завжди зловмисним, є ймовірність того, що програма неоптимізована і має некоректно написані алгоритми, або неправильно використовує ресурси пристрою.

1.5.2 Шляхи вирішення проблеми

Існує ряд рекомендацій, завдяки яким ризик надати обчислювальні потужності свого комп'ютера на користь зловмисників залишається на вкрай низькому рівні. До них належать такі рекомендації як і для шкідливого програмного забезпечення:

– важливо завжди мати антивірус і не забувати оновлювати його бази;

– необхідно користуватися блокувальником реклами;

– не завантажувати файли з неперевірених ресурсів.

Для постійного контролю, рекомендується вивести віджет рівня навантаження процесора на робочий стіл. При Майнінгу через браузер, даний показник може вирости в чотири рази (в порівнянні зі звичайним становищем).

Вкрай важливо звертати увагу на будь-які зміни в поведінці комп'ютера, і ні в якому разі не знижувати концентрації уваги при роботі з заповненням інформації про переказ коштів (особливо стосується довгих адрес гаманців)[7].

Одним із способів протидії кріптоджекінгу є додавання атрибутів Subresource integrity (SRI) до елементів сценарію, що завантажує зовнішні скрипти. Також можна додати механізм Content Security Policy (CSP, політика захисту контенту), за допомогою якого можна захищатися від атак з впровадженням контенту. Однак у таких підходів є свої мінуси - наприклад, якщо вам треба регулярно оновлювати скрипт залежностей. CSP допомагає обмежити завантаження зовнішнього JavaScript на веб-сайт, але він не призначений для забезпечення цілісності сценаріїв, які передбачається завантажити.

Таким чином, реалізувати CSP на основі білого списку - сумнівна ідея, досвідчений зловмисник, ймовірно, все одно знайде спосіб обійти цей механізм, так як він не запобігає ін'єкцію коду з зовнішніх джерел, які, наприклад, можуть розташовуватися на доменах з білого списку.

Рішення проблеми шляхом моніторингу веб-сторінки в реальному часі. Якщо у вас немає гарантованого способу запобігти ін'єкцію виявлення шкідливих програм на сайт, можливо, варто моніторити його впровадження і реагувати в режимі реального часу. Контролюючи DOM і середу JavaScript на предмет будь-ін'єкції, сайт може рапортувати на бекенд, використовуючи механізм webhook. Така схема дозволить виявити будь-які зміни, що допоможе зупинити не тільки відомі, але й 0-day загрози.

Про виявленні аномалії будуть повідомлятися команди безпечників, які зможуть оперативно усунути впроваджений зі шкідливим кодом. Такий підхід вкрай корисний, так як з його допомогою можна визначити, куди саме було

зроблено шкідлива ін'єкція, а потім вжити відповідних заходів для усунення проломи. Як уже було відзначено вище, це дозволить протистояти загрозам нульового дня [8].

Якщо користувач став жертвою кріптоджекінга, наприклад внаслідок атаки шкідливого об'єкта на локальному рівні системи або через браузер, то вручну виявити наслідки вторгнення досить важко. А пошук причин занадто інтенсивної роботи процесора також може бути нелегким завданням. Шкідливі процеси можуть ховатися глибоко в надрах системи або маскуватися під звичайні програми, уникнути видалення. У кріптоджекінга є ще один «бонус»: коли комп'ютер працює на максимальній потужності, він виконує всі операції вкрай повільно, що ще більш ускладнює пошук шкідливих об'єктів в системі. Як і у випадку з іншим шкідливим ПЗ, краще заздалегідь вжити заходи обережності[1].

Один з очевидних варіантів – відключити JavaScript в браузері, що використовується для роботи в Інтернеті. Це відключить можливість тіньового Майнінгу; але мінусом цього методу є обмеження використання в повній мірі потрібних функцій веб-сайтів. Альтернативними шляхами э використання спеціальних програмам, наприклад No Coin або MinerBlock, які блокують активність Майнерів на популярних веб-сайтах. Обидві ці програми мають розширення для браузерів Chrome, Firefox і Opera. Останні версії Opera навіть мають вбудовану функцію NoCoin[1, 4].

1.6 Постановка завдання (на основі першого розділу що буде зроблено у 2 та 3 розділах)

Проаналізувавши інформаційні джерела на тему кріптоджекінга, SQL - ін’єкцій та xss-атак були визначені основні типи цих атак, а також на що вони спрямовуються. Було визначено як саме впливає прихований майнінг на роботу пристроїв, що видає його потенційну присутність. У подальшому ці аспекти будуть використовуватись для аналізу на наявність таких атак.

На основі першого розділу будуть розроблені алгоритми для виявлення розглянутих загроз, які повинні максимально швидко і оптимально виконувати

свої основні функції. У третьому розділі буде продемонстровано роботу готової програми, яка дозволяє проводити моніторинг та аналіз заражених веб-ресурсів, або ресурсів із заздалегіть встановленим кріптоджекером, буде перевірятись вразливість до XSS-атак та SQL -ін'єкцій. Також буде протестовано роботу програми у реальних умовах для складання статистики і вимірювання похибки спрацювання.

2 РОЗРОБКА МОДЕЛЕЙ ВИЯВЛЕННЯ ЗАГРОЗ

2.1 Розробка моделі виявлення вразливості до SQL-ін'єкцій

SQL-ін'єкція – це атака, спрямована на веб-додаток, в ході якої конструюється SQL-вираз з призначеного для користувача введення шляхом простої конкантенації (наприклад, $ query = "SELECT * FROM users WHERE id =". $ _ REQUEST ["id"]) . У разі успіху атакуючий може змінити логіку виконання SQL-запиту так, як це йому потрібно[12].

На рис. 1 показана проста атака з використанням SQL-ін'єкції. З точки зору архітектури користувач при посередництві веб-клієнта взаємодіє з HTTP фронтенда веб-сервера, який, в свою чергу, взаємодіє з бекенд у вигляді SQL-сервера. У ситуації входу користувача в систему цей фронтенд веб-сервера застосовує надану користувачем інформацію при побудові SQL-запиту. Веб-сервер виконує наступну SQL-операцію (в якій uname і pword є вхідними змінними):

```
select * from Users where userid=’uname’ AND password= ‘pword’;
```

Ввівши в цю веб-форму підібрану інформацію, зловмисник може обійти наміри розробника і модифікувати виконуваний запит. У прикладі (рис. 2.1) змінені умови запиту, щоб запросити всі записи. Для використовувався вираз OR (яке завжди має значення true) і за допомогою коментаря деактивується перевірка пароля[13].

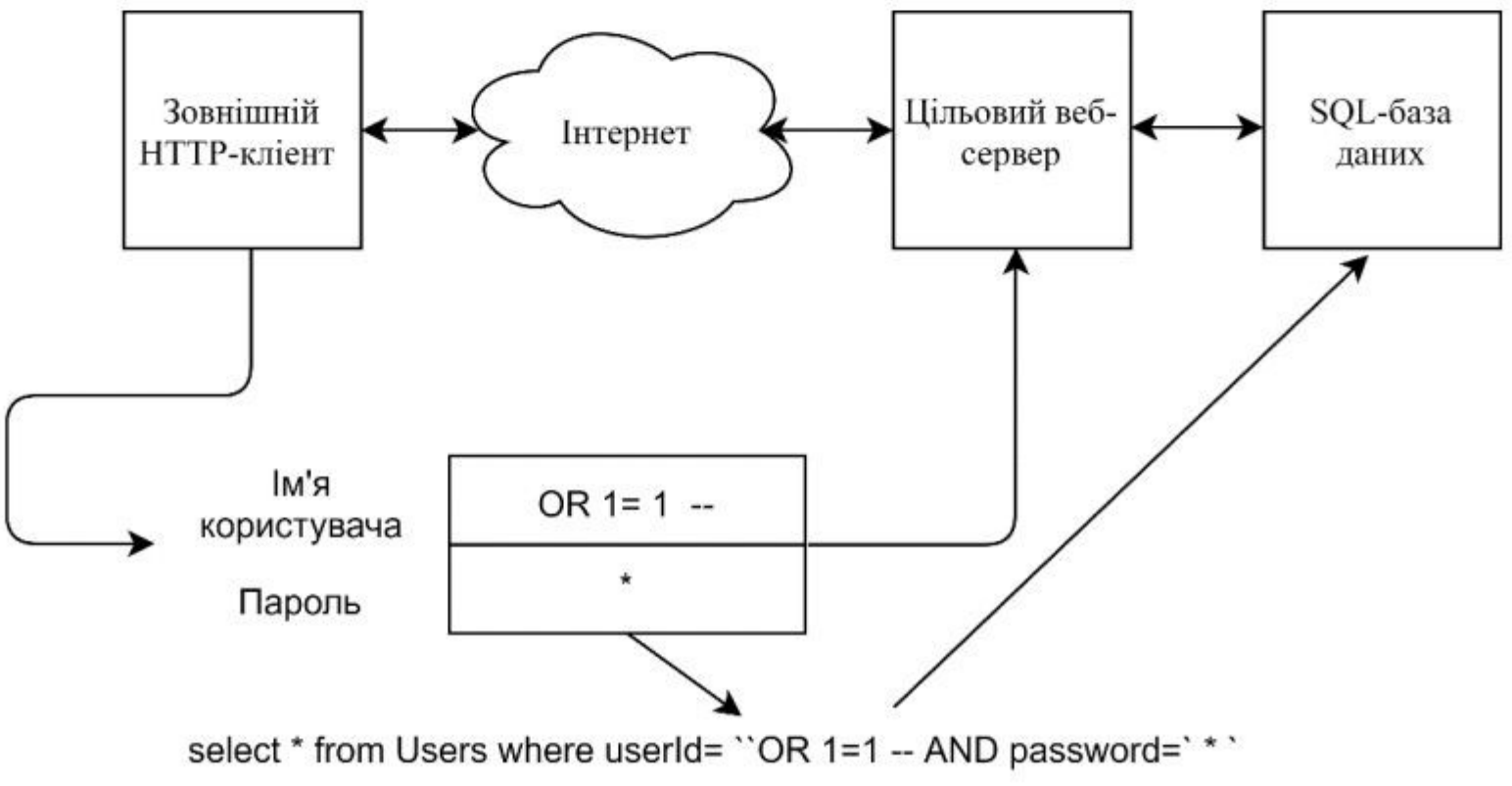


Рисунок 2.1 – Приклад експлуатування вразливості SQL-ін’єкції

Знаючи те, яким чином відбувається SQL-ін'єкції, для перевірки веб-ресурсу на вразливість до цієї атаки, можна її зімітувати, виконавши прості запити, щоб прослідкувати яким чином буде реагувати сервер.

Для перевірки на вразливість до цієї атаки, а також XSS, буде використано такий інтрумент як Selenium WebDriver, який надасть можливість імітувати роботу веб-браузера.

Для перевірки на наявність вразливості на стороні клієнта вводиться посилання на потрібний веб-ресурс та заповняються додаткові поля для тестування. Відправлена форма надходить до серверу, який передає вбудовані або надіслані інструкції до Selenium WebDriver. Згідно з переданими інструкціями буде пройдено три ітерації:

- коректні дані;
- некоректні дані;
- SQL-ін'єкцію.

Метою пройдених ітерацій є їх подальше порівняння.

У випадку коли перша ітерація дорівнює другій, буде зроблено висновок про те, що на даному веб-ресурсі відсутня валідація або вона дуже спрощена. Це свідчить про те, що такий веб-ресурс може бути вразливим до різного типу атак.

Перевірка на подібність другої та третьої ітерації надасть інформацію про вразливість до SQL-ін'єкцій. Якщо сайт по-різному реагує на звичайний набор символів, що не пройшов валідацію та потенційно шкідливий код, це говорить про вразливість до цього типу атак.

За допомогою WebDriver відкривається вікно браузера за отриманим від сервера посиланням, заповнюються усі поля та url, в які потенційно може бути вбудована ін'єкція, коректними даними. Після закінчення зберігається отриманий snapshot та надсилається до сервера, після чого відбувається наступна ітерація. У другому циклі підставляються свідомо некоректні дані, щоб перевірити як веб-ресурс на це відреагує і на яких полях не міститься валідації. Результат відправляється на сервер. На третій ітерації програма

підставляє у потрібні поля рядки, які містять у собі SQL-ін'єкції. Після закінчення третьої ітерації збережений snapshot також відправляється на сервер. На сервері порівнюються усі три результати між собою. Друга, некоректна ітерація порівнюється спочатку із першою, а потом із третьою. Обидва результати порівнянь відправляються на сторону клієнта, де йому відображається інформація щодо сканування. Також пропонується вжити певних заходів проти даної вразливості.

На рисунку 2.2 зображена модель виявлення SQL-ін’єкцій у вигляді схеми.

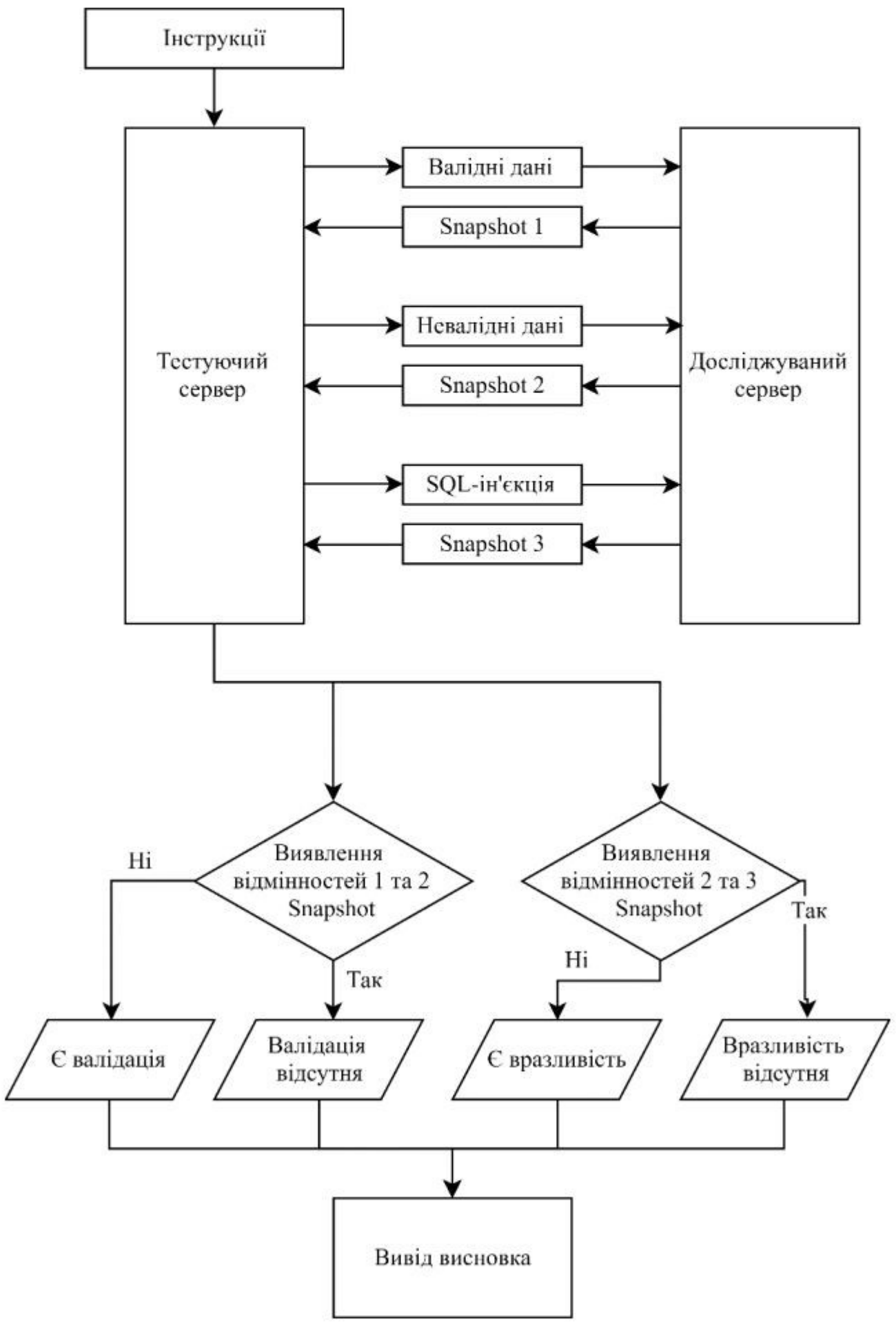


Рисунок 2.2 – Модель виявлення вразливості до SQL-ін’єкцій

Для визначення різниці між Snapshot 1,2,3, був розроблений алгоритм. Вхідними даними є сам Snapshot, який являє собою DOM-дерево досліджуваного сайту, яке представлено у типі даних string.

При реалізації атаки з'являється новий контент, який повертає ін'єкція, це може бути список користувачів, товарів, тощо. У випадку звичайного некоректного запиту, сервер поверне помилку, у більшості випадків 404. З цього виходить, що контент у цих двох випадках буде значно відрізнятись, але буде міститись певна спільна інформація самого веб-ресурсу.

Вхідні дані фільтруються: head, <header>, <footer>, <nav> відсікаються, залишається лише контент ресурсу (рис. 2.3). Для формування множини контент для кожної ітерації зберігається у Set, який являє собою колекцію для зберігання безлічі значень, причому кожне значення може зустрічатися лише один раз.

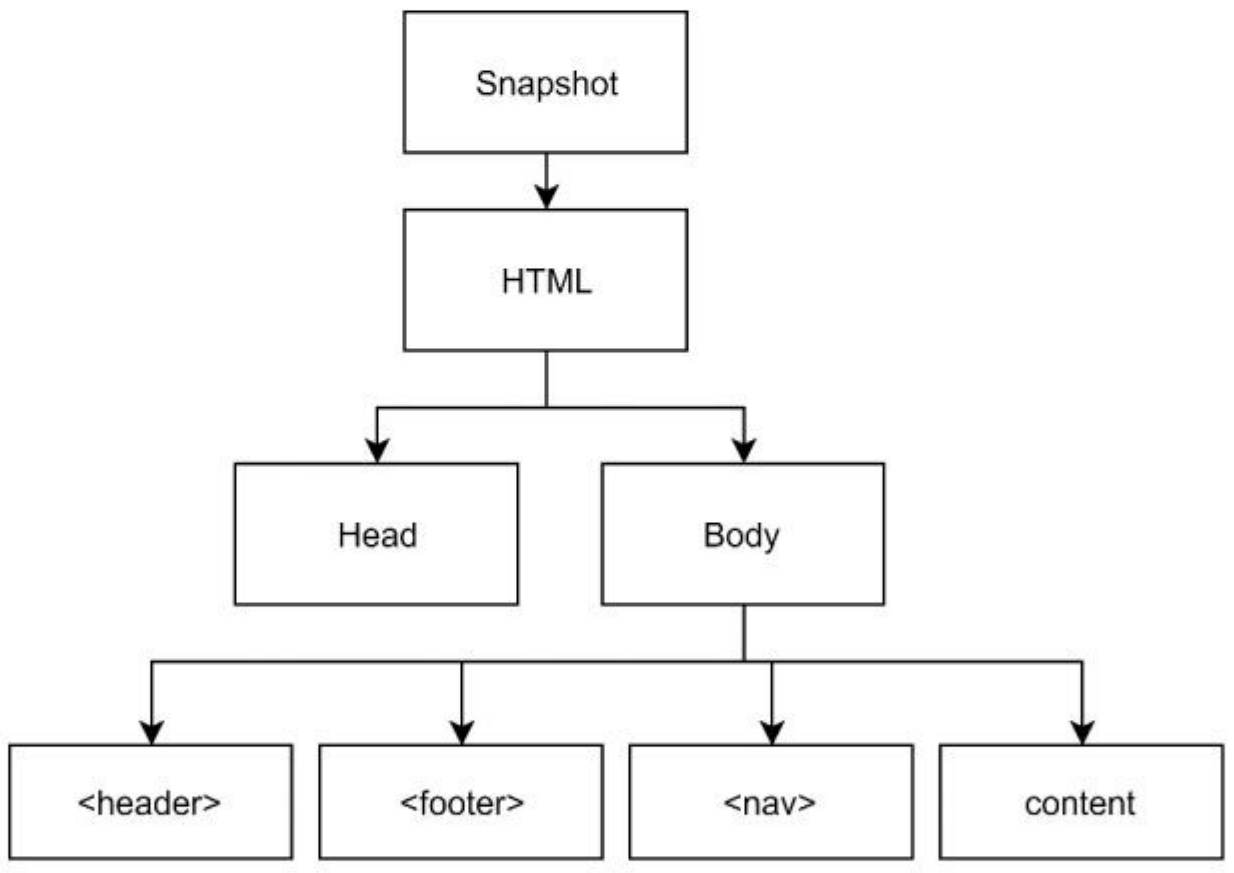


Рисунок 2.3 – Фільтрація DOM-дерева

У відмінності від роботи із звичайними масивами, цей спосіб зберігання надасть більше швидкодії під час перевірки. Множина являє собою набір з DOM-елементів, представлених у вигляді рядків. Наступним етапом йде саме порівняння множин, де різниця (a \ b) (рис 2.4): створить набір, який містить ті елементи множини a, які відсутні у множині b.

Рисунок 2.4 – Множини B – результат коректного запиту, підпадає під множину A – результат виконання SQL-ін’єкції

Для визначення різниці використовується наступний код:

```
const a = new Set([a,b,c]);
const b = new Set([d,c,b]);
const difference = new Set(
[...a].filter(x => !b.has(x)));
```

Результатом його виконання буде різниця між коректним і некоректним запитом, із запитом, який містив SQL-ін’єкцію (рис 2.5).

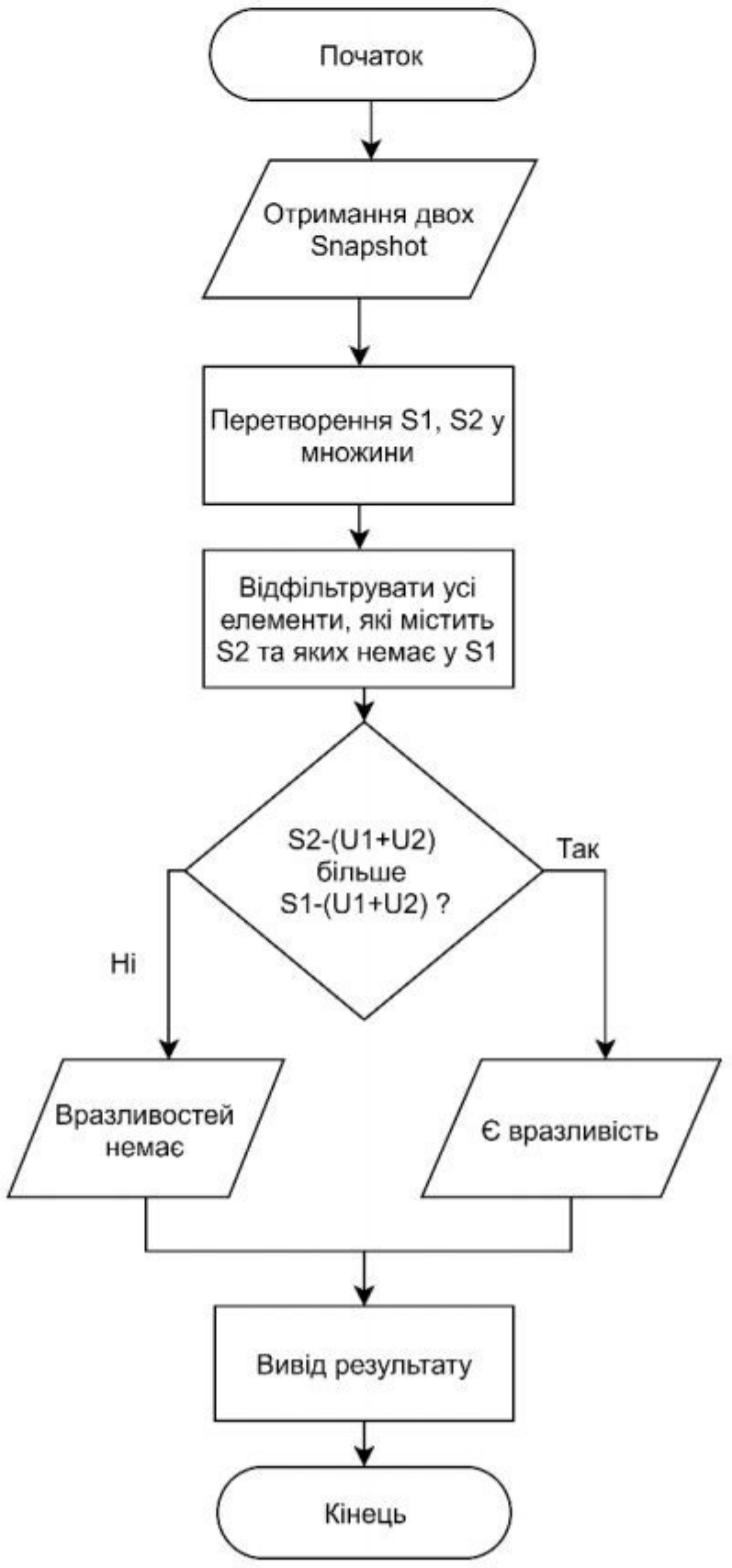


Рисунок 2.5 – Алгоритм пошуку різниці множин

Коректний запит являє собою звичайний запит з параметрами у вигляді: id?=5. Некоректний представлений у вигляді аналогічного запиту, але з довільним набором символів: id?=dRW$awd, та який покликаний спровокувати 404-помилку. Параметри запиту із ін'єкцією мають вигляд: id?=ін'єкція.

Далі отримані DOM-дерева сторінок у результаті запиту із некоректно згенерованим uri-параметром і запиту із SQL-ін'єкцією перетворюються у множини і порівнюються (табл. 2.1), де S1, S2 – snapshot з ін'єкцією і некоректним запитом відповідно, а u1 та u2 – передані uri-параметри у першому і другому запиті. Це робиться з метою, якщо сторінка не буде вразливою і в обох випадках з'явиться помилка 404, тоді довжина множин буде різною у місці, де повідомляється про помилку і вказується некоректно введена адреса.

Таблиця 2.1 – Очікуванний результат порівняння множин

Множина DOM-елементів	SQL-ін'єкція	Некоректний запит	S1-(u+u2) > S2-(u+u2)
Невразливий сайт	S1	S2	Ні
Вразливий сайт	S1	S2	Так

Якщо результат віднімання множини, отриманої з sql-ін'єкції від множини із некоректним uri-параметром буде додатнім, тоді можна затверджувати, що даний веб-ресурс вразливий до даного виду атак.

2.2 Розробка моделі виявлення вразливості до XSS-атак

XSS (міжсайтовий скріптинг) – це тип ін'єкції, який є різновидом атаки «впровадження коду», при якому вбудований на атакованій сторінці шкідливий код, взаємодіє із сервером зловмисника. XSS-атаки трапляються, коли зловмисник використовує веб-додаток для надсилання шкідливого коду, як правило, у вигляді скрипту на стороні браузера, іншому кінцевому користувачеві.

Існує три види такого типу ін'єкцій:

– Reflective – така атака не зберігається на сайті, вона виконується лише в одному запиті і відповіді;

– Stored – такий тип атак зберігає шкідливий код на сервері і вбудовується в сторінки звичайних користувачів;

– Self – ці атаки також не зберігаються на сайті і зазвичай використовуються як частина обману людини з метою запуску XSS їм самим.

За допомогою такої ін'єкції зловмисник може вбудувати будь-який код. Це може бути підроблена копія певного сайту, кейлогер або кража кукі.

<script>window.location='http://site/?cookie='+document.cookie</script>

Для встановлення наявності даної вразливості потрібно зробити перевірку із вбудовуванням простих скріптів на перевіряємі веб-ресурси. Щоб дізнатись про вразливість, буде вбудовуватись код, який міститиме get-запит. У разі вдалої ін’єкції, запит прийде на тестовий сервер.

Для перевірки на наявність вразливості на стороні клієнта вводиться посилання на потрібний веб-ресурс та заповняються додаткові поля для тестування. Відправлена форма надходить до серверу, який передає вбудовані або надіслані інструкції до Selenium WebDriver. У відкрите тестове вікно браузера підставляються скріпти, які надійшли з клієнта або за замовчуванням віддає сервер, якщо на стороні клієнта нічого не було введено. Тестовий сервер очікує отримання get-запиту. Так як різні сервери можуть мати певну затримку або можуть бути завантаженими у даний проміжок часу, запит буде очікуватись 10 секунд. Якщо сервер так і не отримає відповідь за цей час, то як висновок сервер повідомить клієнте про те, що даний веб-ресурс не вразливий до цієї атаки або запропонує повторити процедуру через певний час. Сформовані висновки буде надіслано на сторону клієнта, де він зможе побачити деталі аналізу, а також рекомендації, якщо це буде потрібно.

На рисунку 2.5 зображена модель виявлення XSS-атак у вигляді схеми.

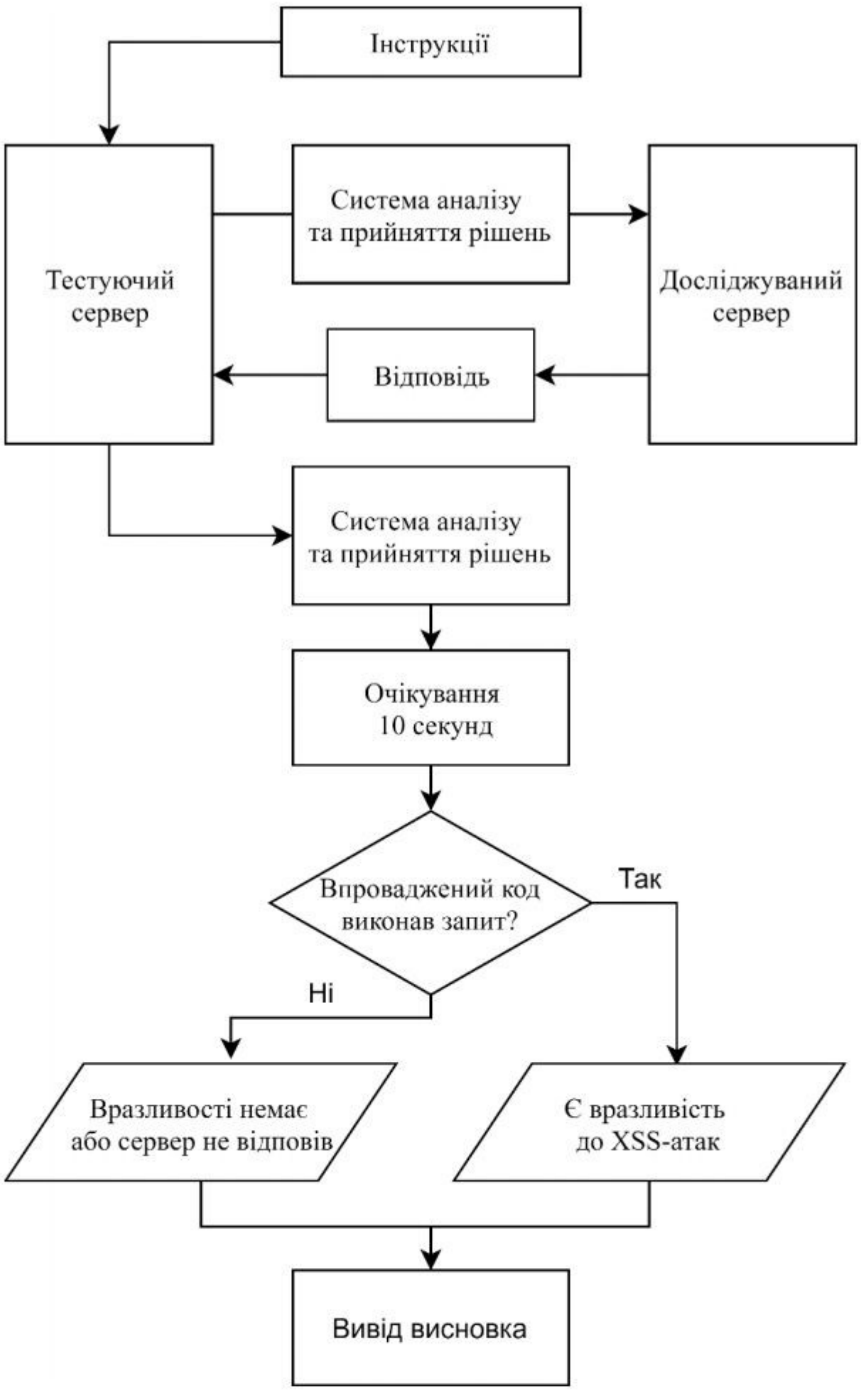


Рисунок 2.5 – Модель виявлення вразливостей до XSS-атак

2.3 Розробка моделі виявлення кріптоджекінга

Кріптоджекінг – це схема використання чужих пристроїв (комп'ютерів, смартфонів, планшетних ПК або навіть серверів) без відома їх власників з метою прихованого видобування криптовалюти.

Існує три види майнінгу:

- майнінг, який використовує ресурси процесора;
- майнінг за допомогою відеокарти;
- ASIC-майнінг.

Найпопулярнішим з цих трьох видів є саме майнінг за допомогою процесора, адже він знаходиться у кожному комп'ютері, смартфоні або навіть в елементах розумного будинку, які останнім часом також масово використовуються для прихованого видобування кріптовалюти.

Частіше за все такі майнери використовуються на сайтах, де регулярно є велика кількість людей. На розшифрування блокчейну йде велика кількість обчислювальних потужностей, що спричиняє її нагрівання та у випадках з комп'ютерами та іншою технікою, працююче на повну потужність охолоджування. У такому випадку можна відкрити системний диспетчер завдань і передивитись усі процеси на комп'ютері. Виявити майнер можливо за великою кількістю споживаних ресурсів. Але сучасні майнери частіше за все приховують себе або вимикаються під час роботи диспетчера завдань. Тому треба аналізувати їх за допомогою спеціальних утиліт.

За допомогою моніторингу Node.js-додатків можна отримати доступ до інформації, яку споживає програма. Це є кращим рішенням у порівнянні вибору процесу з існуючих, адже такий підхід демонструє потрібний у даному випадку процес і те, скільки пам'яті, а також ресурсів процесору він використовує.

Також існують сайти, які містять діаграми, складні анімації, роблять певні математичні обчислення або просто мають неоптимізований код, що спричиняє навантаження на комп'ютер. Але анімації, діаграми та інші обчислення навантажують процесор лише під час самого завантаження сайту або у момент їх створення чи виклику, тому це навантаження не буде сталим.

Для перевірки на наявність вразливості на стороні клієнта вводиться посилання на потрібний веб-ресурс. Відправлена форма надходить до серверу, який запустить Selenium WebDriver за отриманим посиланням[17]. Після повного завантаження сторінки сервер починає записувати дані щодо споживаних ресурсів процесору. Через кожні 3 секунди робиться вимір, який зберігається для подальшого аналізу. Сканування триває 30 секунд, щоб виключити похибку на завантаження інших скриптів, анімацій, графіків. Після чого усі отримані дані аналізуються за алгоритмом (рис 2.6).

Рисунок 2.6 – Алгоритм порівняння значень із пороговим

Встановлюється мінімальне значення у 40%. Якщо перший вимір більше 40, то він порівнюється з усіма подальшими вимірами. У випадку, коли перший вимір меньший за 40%, то до нього додається це значенгя. Таким чином задається мінімальний показник у 40%, який порівнюється із 2,3,4,5 вимірами. Тоді програма попередить користувача про велику ймовірність наявності майнера.

На рисунку 2.7 зображена модель виявлення кріптоджекерів у вигляді схеми.

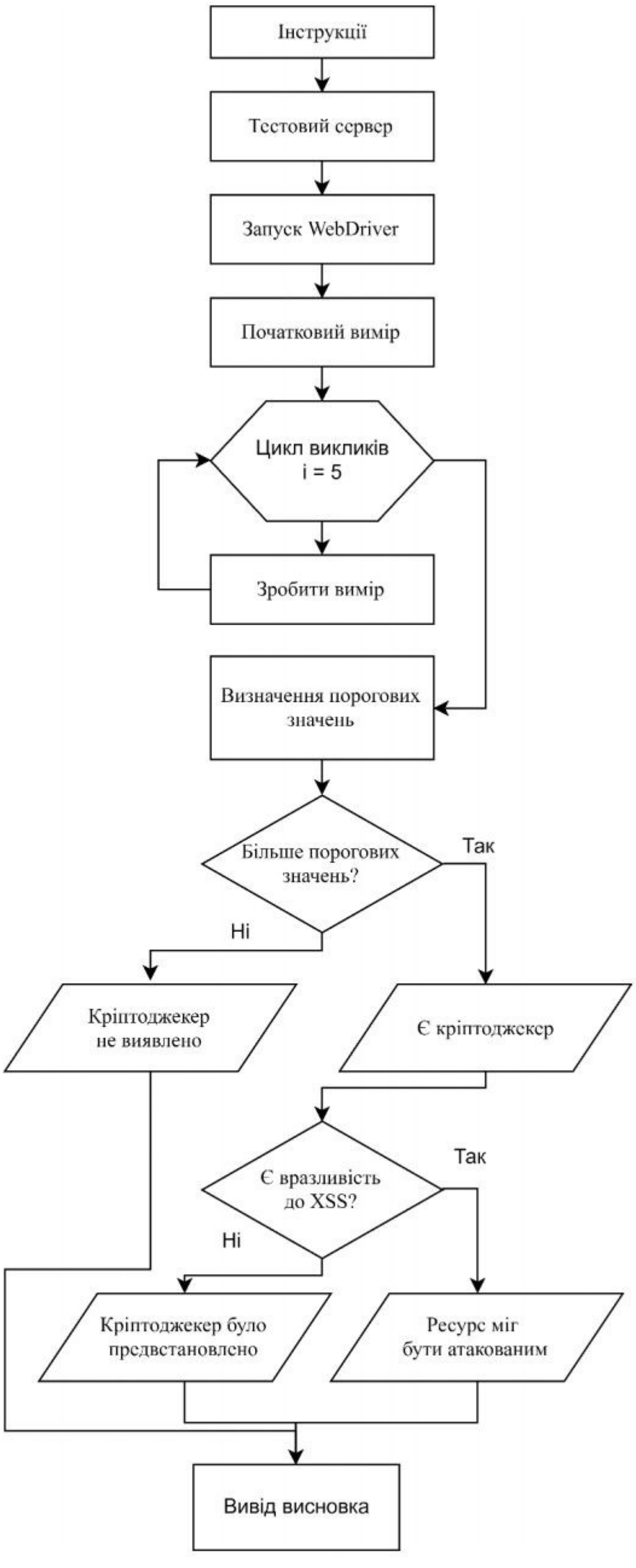


Рисунок 2.7 – Модель виявлення кріптоджекерів

Також у разі позитивного результату після проходження сканування на вразливість XSS-атак, збільшується ймовірність того, що високе навантаження

створює саме кріптоджекер, тому що у зловмисника була можливість вбудувати свій шкідливий код.

У даному розділі розглянуто особливості кожної із загроз і створено моделі для виявлення sqli-ін'єкцій, xss-атак і кріптоджекерів. Також відповідно для кожної моделі розроблений алгоритм прийняття рішення.

У наступному розділі програмно реалізовані описані моделі і розроблено програмний додаток, який виконує сканування веб-ресурсів на наявність описаних вище загроз.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

Для розробки програмного засобу використовувався такий редактор коду як Visual Studio Code. Плюсами цього редактору є зручний інтерфейс та велика кількість різних додатків, які можна завантажити для зручності написання коду. Також цей редактор підтримує TypeScript, JavaScript, Node.js і Mono.

Для розробки програмного засобу була обрана мова JavaScript. Ця мова є дуже поширеною та заточеною більш під веб-додатки, але також завдяки їй можна писати сервери, мобільні та десктопні додатки.

Програма складається з серверної частини та клієнтської. Для написання серверної частини був обраний Node.js, а для клієнтської – React.js, який використовувався у парі з Electron для запуску у якості окремої програми. Electron – фреймворк для створення десктопних програм, з використанням JavaScript, HTML та CSS.

3.1 Структура програмного засобу

Програмний засіб складається з двох частин: сторона сервера, яка виконує сканування, приймає рішення та сторона клієнта з інтерфейсом, яка відображає кінцевий результат, який приходить із сервера.

У корні проекту міститься файл package.json, який зберігає список усіх залежностей для пакету npm, до якого входять підключені бібліотеки під час розробки. Папка node_modules зберігає усі бібліотеки, які використовуються програмою для роботи і розробки.

Серверна частина знаходиться у корені програми, а клієнтська зберігається у папці front, яка також містить package.json і node_modules для своєї роботи.

3.1.1 Клієнтська частина

Клієнтська частина зберігається у папці front. Тут зберігається: public, src, node_modules, pacage.json, а також інші допоміжні файли.

У папці public зберігається основний index.html, який є основною точкою входу для react-додатку.

Папка src містить у собі App.js – основний файл, до якого підключаються компоненти, а також electron-starter.js, який встановлює точку доступу для index.html та створює вікно для запуску додатку electron. Для під'єднання react, використовується electron-wait-react.js, який вказує порт прослуховування та створює з'єднання між electron та react.

Також у src міститься папка Components, у якій є StartPage.js – головний клас програми, до якого підключаються усі додаткові компоненти, а також який виконує запити для сканування на серверну частину. LineGraph.js – клас, який будує графік навантаженності для результатів сканування на наявність кріптоджекерів. StartPage.css описує стилі для відображення клієнтської частини.

3.1.2 Серверна частина

Головним файлом серверної частини є index.js, в якому використовується бібліотека Selenium WebDriver для Node.js [17], а також містить усі ендпоінти та вказується порт для прослуховування. Також цей файл містить логіку для виявлення у веб-ресурса наявності вразливості до XSS-атаки.

У головному файлі містяться три ендпоїнта для надсилання на них post-запитів. Посилання http://localhost:4231/detect_miner викликає перевірку на наявність кріптоджекера і по завершенню сканування відправляє сформовані дані на клієнтську частину. Посилання http://localhost:4231/check_sql викликає порівняння DOM-дерев для визначення вразливості до SQL-ін'єкцій. Для визначення вразливості до XSS-атак, використовується ендпоїнт http://localhost:4231/check_xss.

Файл chromedriver.exe – допоміжний файл, який використовує Selenium WebDriver для запуску емулятора Google Chrome.

Файл compareSnapshots.js містить у собі функції для запуску драйвера, а також порівняння snapshots. diffBetweenSnapshots.js – виконує порівняння між snapshots, які на нього надходять, розбиває їх на множини і перевіряє різницю

між ними. Файл testCPU.js створює виміри для видачі результату під час сканування на наявність кріптоджекера.

3.2 Тестування програмного засобу

Програма передбачає сканування обраного ресурсу за трьома критеріями: вразливість до XSS-атак, SQL-ін'єкцій та наявності прихованих майнерів.

Головне вікно програми містить поле для введення посилання і кнопку «Сканувати».

Робота додатку починається з введення посилання перевіряємого ресурсу у поле пошуку (рис. 3.1).

Рисунок 3.1 – Вигляд головного вікна програми до початку сканування

Після чого для відправлення форми на сервер, треба натиснути кнопку «Сканувати», яка викликає функцію scanForVulnerabilities. Ця функція містить у собі три post-запити по кожному для окремого сканування. Також виклик функції відкриє вікно, яке буде відображати процес сканування (рис. 3.2), вже просканові вразливості і результат щодо вразливості.

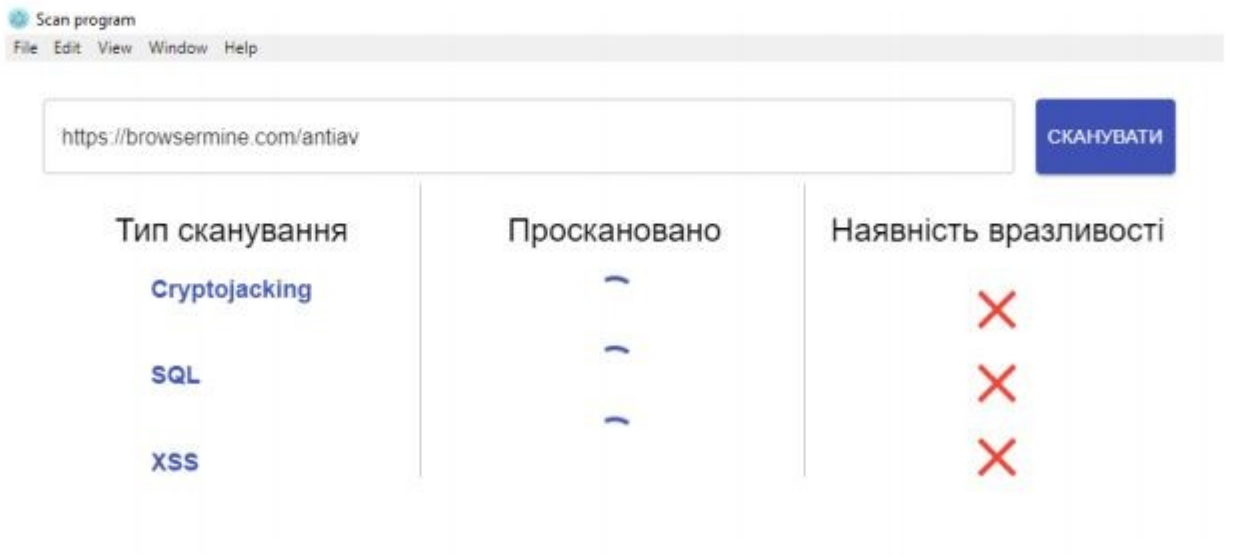


Рисунок 3.2 – Вікно результатів сканування

Першим робиться запит на перевірку кріптоджекерів, тому що ресурс відкриється вперше і у разі наявності кріптоджекера буде видно різкий стрибок навантаженості. За допомогою Node.js os і методу os.cpus(), який повертає масив об'єктів, які містять інформацію про кожне встановлене ядро / CPU.

Використовуючі власні методі os можна вирохувати скільки навантаженності спричиняє робота сервера. Кожні 3 секунди робиться вимір вільного місця на ядрах процесора osu.cpu.free(). Після проведення шести вимірів масив навантажень відправляється на сторону клієнта. По отриманим точкам будується графік для наглядного сприймання змін навантаження, а також робиться висновок щодо наявності кріптоджекера: у випадку середнього значення навантаженності у розмірі 70% і більше, буде заявлено про потенційну наявність кріптоджекера.

На рисунку 3.3 зображено просканований сайт на наявність кріптоджекера. Середня навантаженність склала – 28%. На графіку видно, що основна навантаженність була спочатку, коли сторінка тільки завантажується. Програма видала негативний результат щодо наявності загрози.

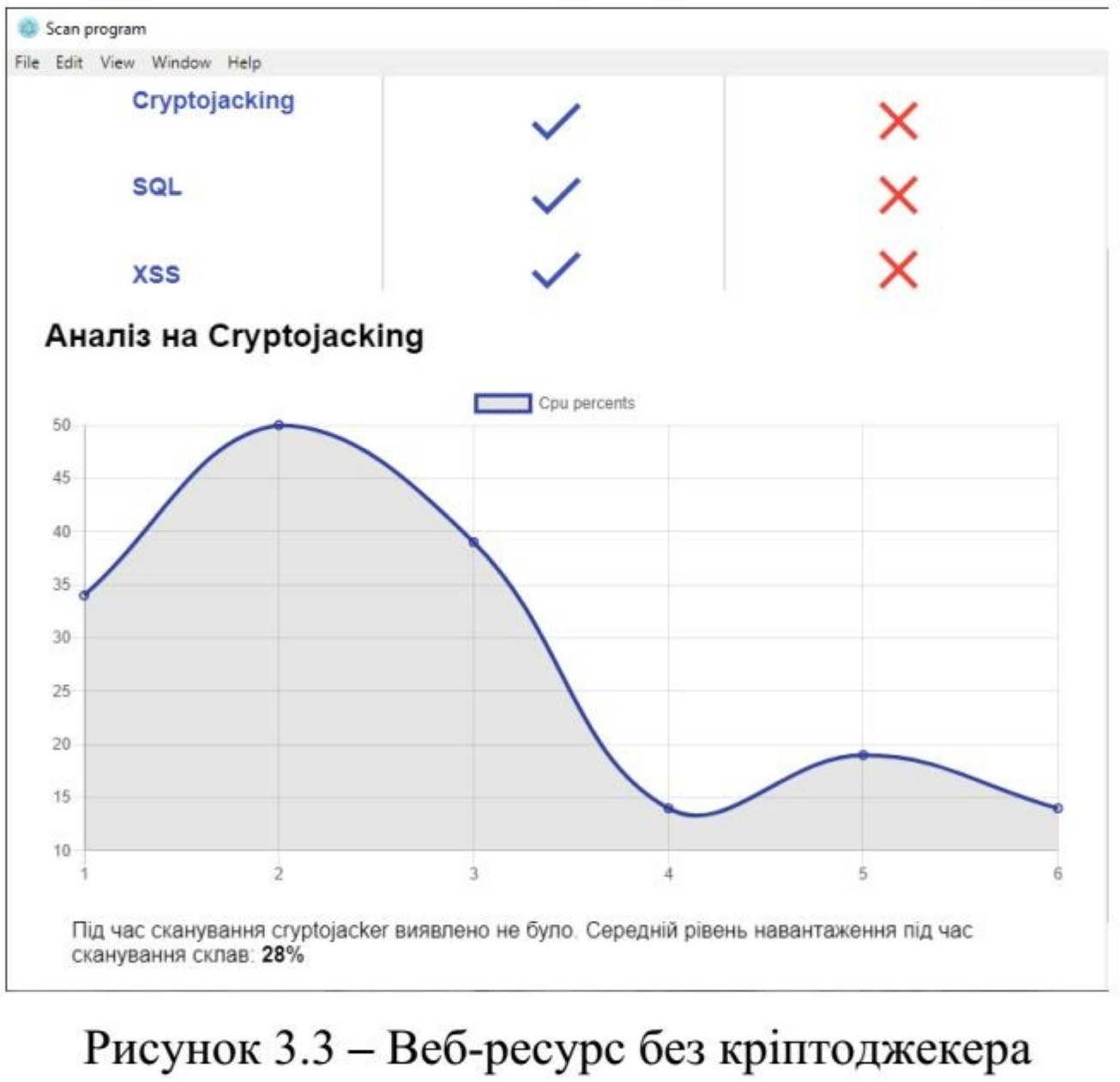


Рисунок 3.3 – Веб-ресурс без кріптоджекера

На рисунку 3.4 зображено сайт, який використовується для майнінгу кріптовалюти. Він використовує ті ж самі алгоритми для видобування валюти, що й приховані, тому він підходить для перевірки роботи методу. По закінченню сканування середній результат складав 79%. Початковий результат виміру складав 35%, другий збільшився д 90%, а 3,4,5 були по 95-100%. Останній вимір робиться після закриття WebDriver, з чого можна побачити

різке зменшення навантаження до 15%, яке було сталим впродовж попередніх вимірів. Програма видала позитивний результат про наявність потенційного кріптомайнера.

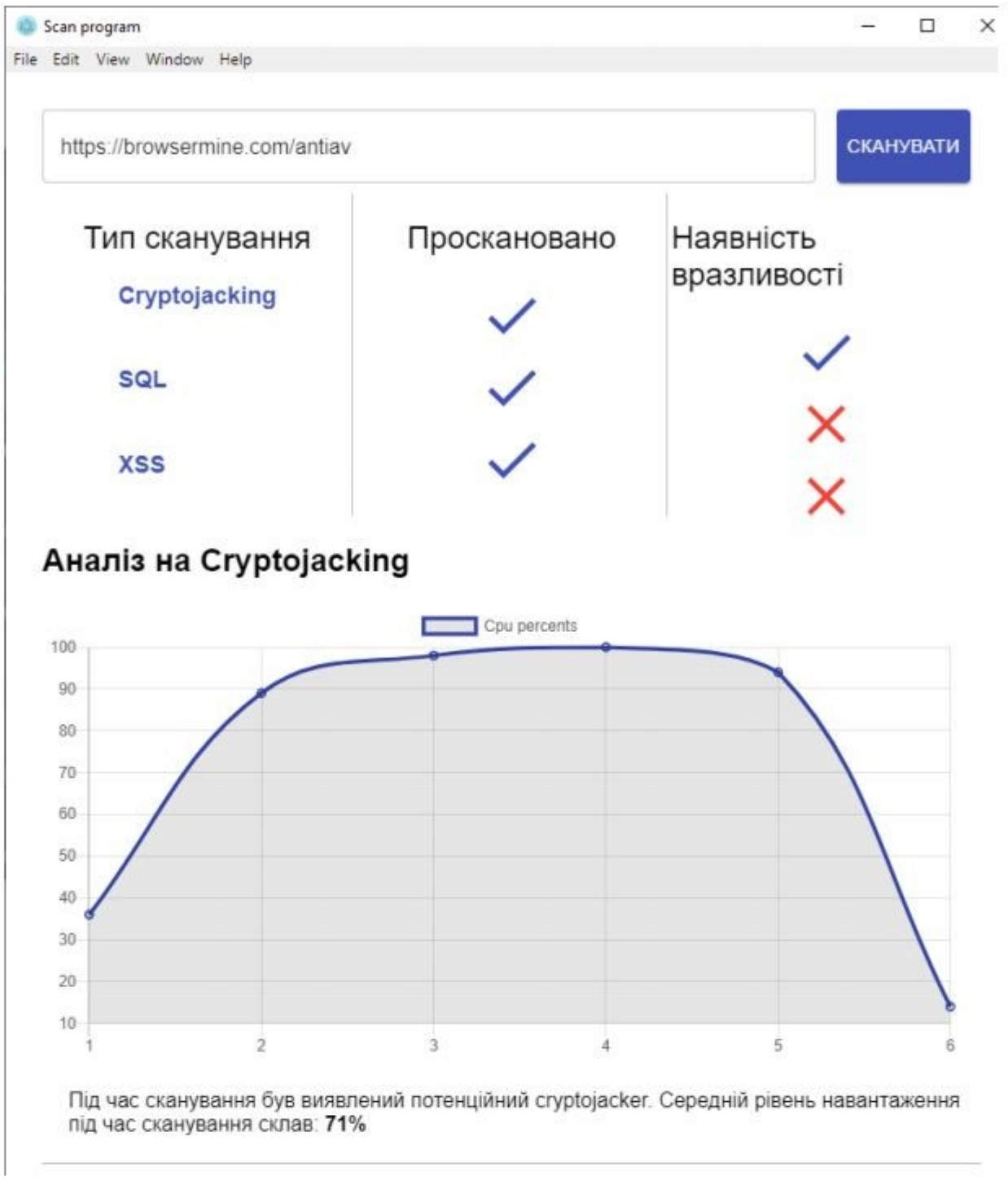


Рисунок 3.4 – Сканування веб-ресурсу, якій містить кріптомайнер

Для сканування SQL-ін'єкцій відправляється post-запит – /check_sql, у якості параметра передається саме посилання на потрібний ресурс. У першу чергу викликається асинхронна функція driver, яка відкриває вікно браузера за допомогою WebDriver. Далі отримується посилання на веб-ресурс, яке було надіслано зі сторони клієнта. На основі отриманого посилання формується urlToCheck – посилання на веб-ресурс із SQL-ін'єкцією в uri-параметрі та invalidUrl – посилання на той самий ресурс, але яке має в uri-параметрі довільний набір символів, згенерований функцією genUri.

У випадку з ін'єкцією виходить https://demo.testfire.net?id=1 or 1=1, де ?id=1 or 1=1 є самою ін'єкцією і запис 1=1 поверне true. Вразливість до таких атак пояснюється тим, що серверні розробники не екранують отримані дані від

користувача, тому такий запит поверне дані, навіть якщо доступ до них закритий.

У випадку із некоректним запитом виходить таке посилання: https://demo.testfire.net?id= tHSRoqH!Ix:9!q6, яке спровокує помилку 404.

Отриманні DOM-дерева з обох запитів фільтруються за допомогою функцій defaultResult і injectionResult відповідно. Результати цих функцій поміщаються у множину Set, після чого порівнюються за допомогою функції difference, відфільтрована різниця зберігається в окремій множині. Якщо довжина difference буде більшою defaultResult не менше ніж на 1%, тоді результат про наявність вразливості буде позитивним.

На рисунку 3.5 зображено сканування на наявність даної вразливості сайту, який не має вразливості.

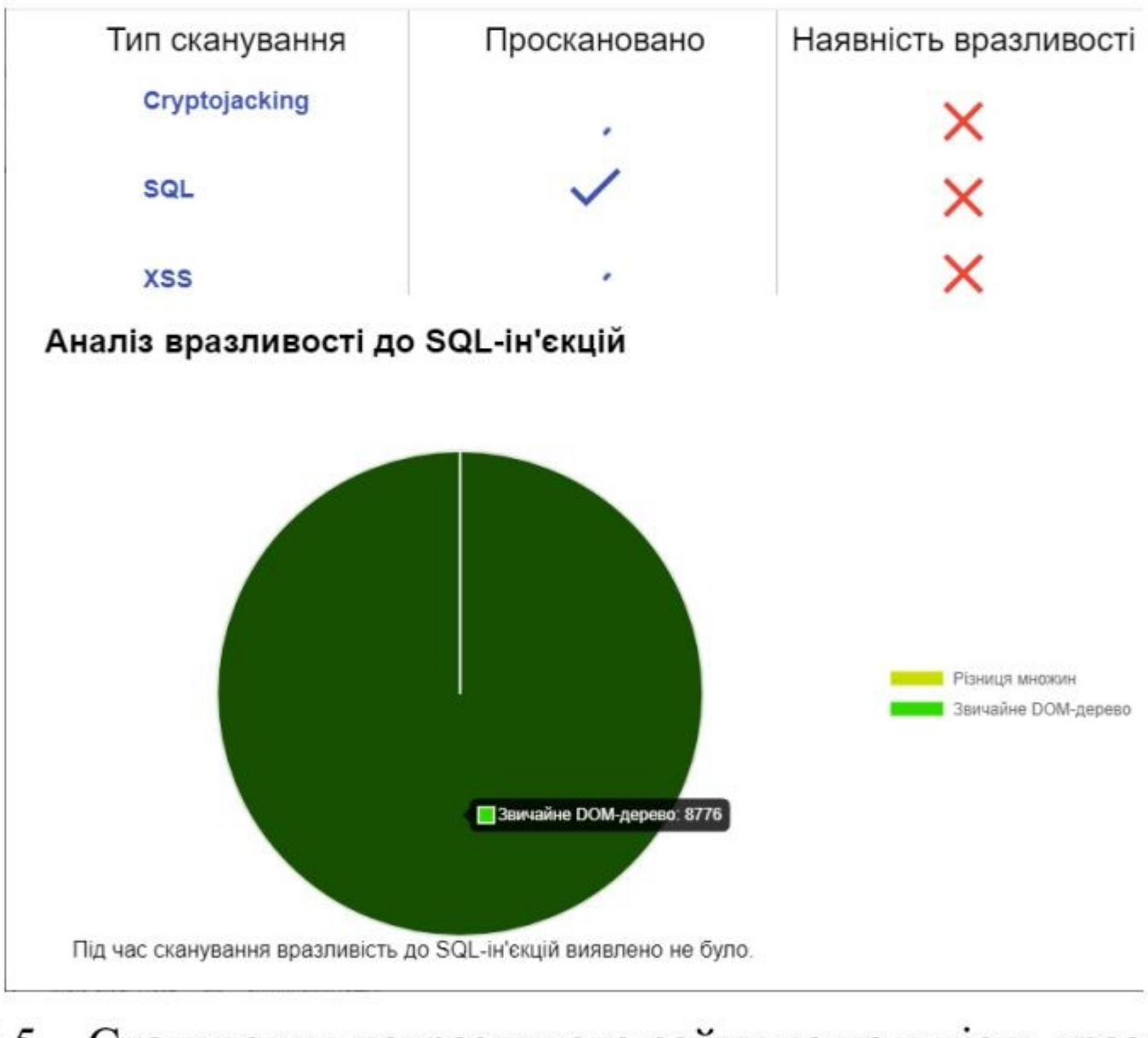


Рисунок 3.5 – Сканування невразливого сайту на наявність вразливості до SQL-ін’єкції

Як видно з рисунку, обидві множини рівні і накладаються одна на одну. Це підтверджує той факт, що вразливості немає.

На рисунку 3.6 зображено просканований сайт, якій містить вразливість до ін’єкцій.

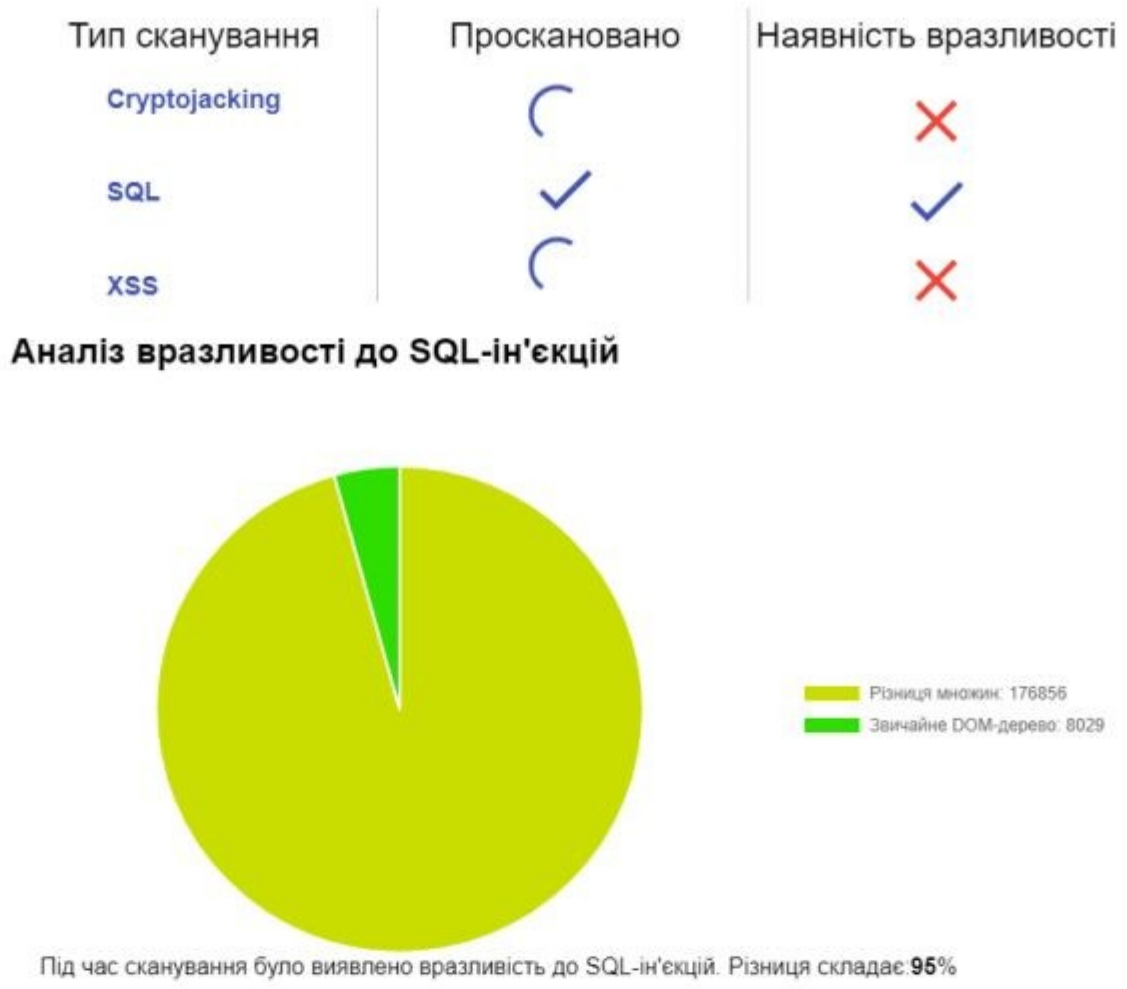


Рисунок 3.6 – Сканування вразливого до ін’єкцій сайту

З рисунку видно, що розмір довжина множини у випадку помилки становить 8029 і займає малу частину графіку, тоді як різниця складає 176856 – 95%. Це свідчить про те, що ін’єкція повернула інформацію із бази даних.

Для сканування XSS-атак відправляється post-запит – /check_xss, у якості параметра передається саме посилання на потрібний ресурс. У першу чергу викликається асинхронна функція driver, яка відкриває вікно браузера за допомогою WebDriver. Далі отримується посилання на веб-ресурс, яке було надіслано зі сторони клієнта.

Далі формується xssFetch – XSS-ін’єкція, яка містить у собі <script> із fetch-запитом, в тілі якого є звернення до document.cookie, localStorage та sessionStorage, для перевірки наявності в них даних. Після завантаження сторінки спрацьовує цикл, який проходить по кожному зверненню і повертає true, якщо містилась якась інформація або false, якщо інформації не було (рис. 3.7). Якщо веб-ресурс не вразливий до цієї атаки, то з’явиться негативне повідомлення в наявності вразливостей і додаткового вікна нижче не з’явиться.

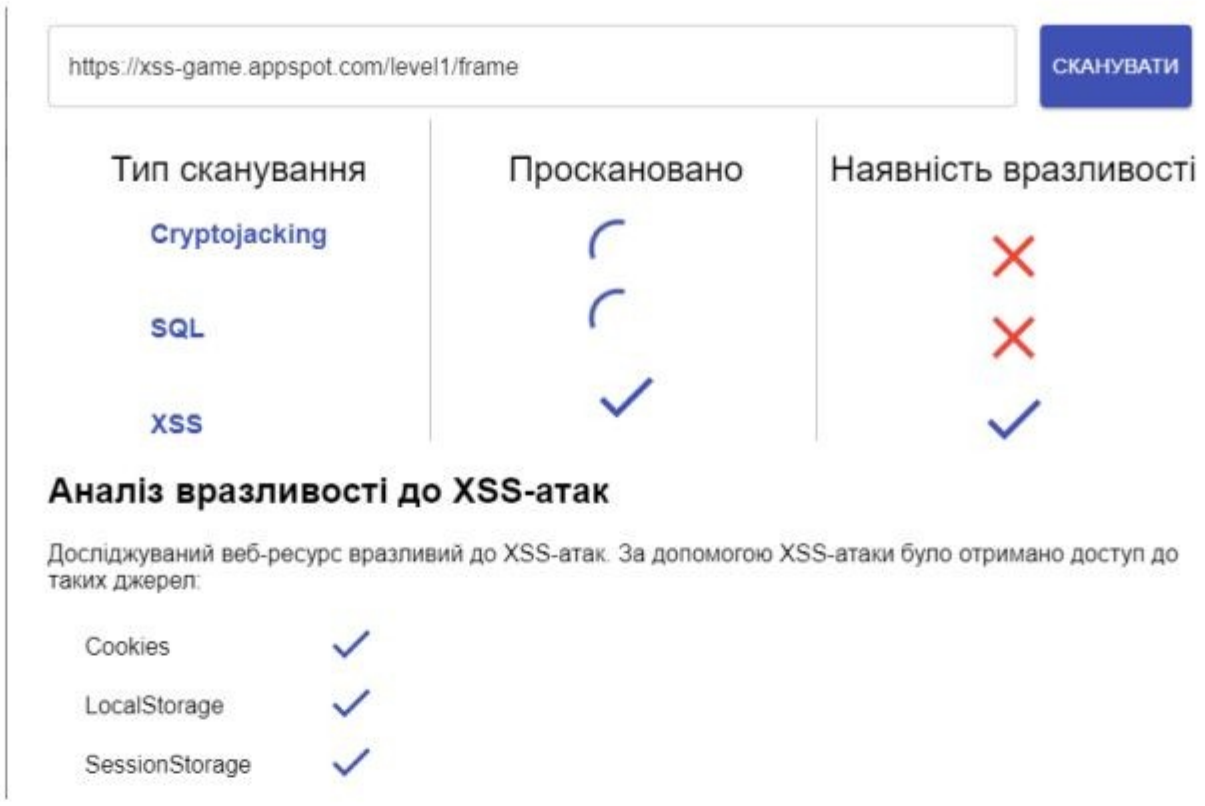


Рисунок 3.7 – Сканування вразливого до XSS-атак ресурсу

На рисунках були наведені приклади реагування на вразливі сайти, які використовуються для тестування описаних загроз. Були проведені додаткові сканування на різних веб-ресурсах для складання статистики і визначення ймовірності похибки (табл. 3.1).

Таблиця 3.1 – Складання вибірки для визначення похибки визначення вразливості до SQL-ін’єкцій

Вразливість / Веб-ресурс	Є вразливість до SQL-ін’єкцій	Вразливістсь було визначено
https://altwall.net/index.php?id=review	Ні	Ні
http://www.smtmax.com/category.php	Так	Так
http://www.programmallp.it/index.php	Ні	Ні
http://c4nholidays.com/deal-details.php	Так	Так
https://www.tcodesearch.com	Ні	Ні
http://www.sfgames.ru/gameS.php	Ні	Ні
http://mtbstickers.ru/index.php	Так	Ні
http://musculoskeletalsociety.in/verify.php	Так	Так
http://www.konghuaschool.org	Ні	Ні
https://arstechnica.com	Ні	Ні
https://stackoverflow.com	Ні	Ні

В таблиці 3.1 був наведений список вразливих і невразливих сайтів. З усіх лише один був визначений некоректно, тому що він вертав довільну кількість даних для кожного запиту із ін'єкцією, що іноді спричиняло від'ємні результати. З урахуванням цього, ймовірність правильної відповіді, виходячі з складеної статистики – 91%.

Таблиця 3.2 – Складання вибірки для отримання похибки визначення вразливості до XSS-атак

Вразливість / Веб-ресурс	Є вразливість до SQL-ін'єкцій	Вразливістсь було визначено
https://altwall.net/index.php?id=review	Ні	Ні
http://www.smtmax.com/category.php	Ні	Ні
http://www.programmallp.it/index.php	Ні	Ні
http://c4nholidays.com/deal-details.php	Ні	Ні
https://demo.testfire.net	Так	Так
http://www.sfgames.ru/gameS.php	Ні	Ні
http://mtbstickers.ru/index.php	Ні	Ні
https://xss-game.appspot.com	Так	Так
http://www.konghuaschool.org	Ні	Ні
https://arstechnica.com	Ні	Ні

З перевірених сайтів усі 10 були коректно визначені на наявність або відсутність вразливості до XSS-атак.

Таблиця 3.3 – Складання вибірки для отримання похибки визначення кріптоджекера.

Вразливість / Веб-ресурс	Є вразливість до SQL-ін'єкцій	Вразливістсь було визначено
https://altwall.net/index.php?id=review	Ні	Ні
https://www.ukrinform.ru	Ні	Ні
https://browsermine.com/antiav	Так	Так
https://predictoria.com	Ні	Ні
https://stackoverflow.com	Ні	Ні

http://c4nholidays.com/deal-details.php	Hi	Hi
https://freebitco.in	Так	Так
https://memoriestore.com	Hi	Hi
https://www.youtube.com	Hi	Hi
https://www.google.com	Hi	Hi
https://webmining.co	Так	Так
https://delo.ua	Hi	Hi
http://www.konghuaschool.org	Hi	Hi
https://www.twitch.tv	Hi	Так
https://telegra.ph	Hi	Hi

З усіх ресурсів лише один був визначено некоректно – twitch.tv. Даний веб-ресурс являє собою платформу для трансляцій ігор у реальному часі, що робить його ресурсопотребуємим. Саме через це алгоритм дав некоректне рішення. Ймовірність коректної відповіді, виходячі із статистики – 94%.

У даному розділі був розроблений програмний засіб відповідно розробленим у другому розділі алгоритмам. Роботу програми було протестовано на сайтах, які заздалегіть мали потрібні вразливості, отримані результати були порівняні з невразливими сайтами.

У наступному розділі буде розрахована ціна розробленого програмного продукту.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Визначення рівня комерційного потенціалу розробки моделі і засобу виявлення вразливостей у веб-додатках

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки моделі і засобу виявлення вразливостей у веб-додатках. В результаті оцінювання можна буде зробити висновок щодо напрямів (особливостей) організації подальшого її впровадження з врахуванням встановленого рейтингу.

Для проведення технологічного аудиту залучимо 3-х незалежних експертів. У нашому випадку такими експертами будуть керівник магістерської роботи та провідні викладачі випускової та споріднених кафедр.

Оцінювання комерційного потенціалу розробки моделі і засобу виявлення вразливостей у веб-додатках будемо здійснювати за 12-ю критеріями згідно рекомендацій.

Результати оцінювання комерційного потенціалу розробки моделі і засобу виявлення вразливостей у веб-додатках заносимо до таблиці 1.1.

Таблиця 4.1. - Результати оцінювання комерційного успіху розробки моделі і засобу виявлення вразливостей у веб-додатках

Критерії	Експерти		
	к.т.н. доц. Дудатьєв А. В.	к.т.н., ст. викл Лукічов В. В.	к.т.н., доцент Войтович О.П.
	Бали, виставлені експертами		
1	1	2	3
2	3	3	2
3	4	4	3
4	2	2	1
5	3	2	1
6	4	3	4
7	4	4	3

8	3	3	3
9	3	2	3
10	2	2	3
11	3	4	3
12	2	3	2
Сума балів	33	31	31
Середньоарифметична сума балів, СБ	28		

За даними таблиці 4.1 робимо висновок щодо рівня комерційного потенціалу розробки моделі і засобу виявлення вразливостей у веб-додатках. При цьому користуємося рекомендаціями, наведеними в таблиці 4.2.

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 50	Високий

Таким чином, робимо висновок, щодо рівня комерційного потенціалу нашої розробки моделі і засобу виявлення вразливостей у веб-додатках – середній.

4.1.1 Визначення рівня якості розробки моделі і засобу виявлення вразливостей у веб-додатках.

Оцінювання рівня якості розробки моделі і засобу виявлення вразливостей у веб-додатках проводиться з метою порівняльного аналізу і визначення найбільш ефективного, з технічної точки зору, варіанта інженерного рішення.

Рівень якості – це кількісна характеристика міри придатності певного виду продукції для задоволення конкретного попиту на неї при порівнянні з відповідними базовими показниками за фіксованих умов споживання.

Абсолютний рівень якості розробки моделі і засобу виявлення вразливостей у веб-додатках знаходимо обчисленням вибраних для її вимірювання показників, не порівнюючи їх із відповідними показниками аналогічних виробів. Для цього необхідно визначити зміст основних функцій, які повинні реалізовувати розробка, вимоги замовника до неї, а також умови, які характеризують експлуатацію, визначають основні параметри, які будуть використані для розрахунку коефіцієнта технічного рівня виробу. Система параметрів, прийнята до розрахунків, повинна достатньо повно характеризувати споживчі властивості інноваційного товару (його призначення, надійність, економічне використання ресурсів, стандартизація тощо).

Далі визначаємо величину параметрів якості в балах та встановлюємо граничні його значення (кращі, гірші, середні). Всі ці дані для кожного параметра заносимо в табл. 4.3.

Таблиця 4.3 – Основні параметри моделі і засобу виявлення вразливостей у веб-додатках

Параметри	Абсолютне значення параметра			Коефіцієнт вагомості параметра
	Краще +5…+4	Середнє +3	Гірше +1…+2	
Швидкодія			2	0,2
Точність аналізу		3		0,2
Доступність	5			0,4
Імовірність похибки		3		0,2

Із врахуванням коефіцієнтів вагомості відповідних параметрів можна визначити абсолютний рівень якості інноваційного рішення за формулою:

$$К_{я.а.} = \sum_{i=1}^{n} Рні \cdot аі, \quad (4.1)$$

де Рні – числове значення і-го параметра інноваційного рішення, n – кількість параметрів інноваційного рішення, що прийняті для оцінювання, аі – коефіцієнт вагомості відповідного параметра (сума коефіцієнтів вагомості всіх параметрів повинна дорівнювати 1).

Отже, абсолютний рівень якості моделі і засобу виявлення вразливостей у веб-додатках становитиме – 3,6 бали.

Одночасно визначаємо відносний рівень якості моделі і засобу виявлення вразливостей у веб-додатках, що виробляється (проектується), порівнюючи її показники з абсолютними показниками якості найліпших вітчизняних та зарубіжних аналогів (товарів-конкурентів) (табл. 4.4).

Таблиця 4.4 – Основні параметри моделі і засобу виявлення вразливостей у веб-додатках та товару-конкурента

Параметри	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (конкурент)	Новий		
Швидкодія	1	2	2	0,2
Точність аналізу	85%	90%	1,105	0,2
Доступність	1	3	3	0,4
Імовірність похибки	10-30%	10-20%	1,5	0,2

Відносний рівень якості методу та засобу завадостійкого розподілу секрету визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} qi \cdot ai, \quad (4.2)$$

За розрахунками відносний рівень моделі і засобу виявлення вразливостей у веб-додатках становитиме – 2,105. Це означає, що наша розробка краща за якістю на 71% від товару-аналога.

4.1.2 Визначення конкурентоспроможності розробки моделі і засобу виявлення вразливостей у веб-додатках

У найширшому розумінні конкурентоспроможність товару – це можливість його успішного продажу на певному ринку і в певний проміжок часу. Водночас конкурентоспроможною можна вважати лише однорідну продукцію з технічними параметрами і техніко-економічними показниками, що ідентичні аналогічним показникам уже проданого товару. Для того, щоб високоякісний товар був одночасно і конкурентоспроможним, він має відповідати критеріям оцінювання споживачів конкретного ринку в конкретний час.

Дані для розрахунку загального показника конкурентоспроможності розробки необхідно занести до таблиці 4.5.

Таблиця 4.5 – Нормативні, технічні та економічні параметри моделі і засобу виявлення вразливостей у веб-додатках і товару-конкурента

Параметри	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (конкурент)	Новий		
Швидкодія	1	2	2	0,2
Точність аналізу	85%	90%	1,105	0,2
Доступність	1	3	3	0,4

Імовірність похибки	10-30%	10-20%	1,5	0,2
Ціна за продукт, тис. грн.	14500	3200	0,22	-

Загальний показник конкурентоспроможності розробки (К) з урахуванням вищезазначених груп показників визначаємо за формулою:

$$К = \frac{Іт.п.}{Іе.п.} = \frac{2,105}{0,22} = 9,56, \quad (4.3)$$

де Іт.п. – індекс технічних параметрів (відносний рівень якості інноваційного рішення); Іе.п. – індекс економічних параметрів.

$$Іе.п. = \frac{РНеі}{РБеі} = \frac{3200}{14500} = 0,22, \quad (4.4)$$

де РНеі, РБеі – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

Згідно розрахунків загальний показник конкурентоспроможності – 2,23 . Це означає, що наша розробка моделі і засобу виявлення вразливостей у веб-додатках більш конкурентна у 9,5 разів від товару-аналога.

4.2 Прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи

4.2.1 Розрахунок витрат, що стосуються виконавців розробки моделі і засобу виявлення вразливостей у веб-додатках

Основна заробітна плата кожного із розробників (дослідників) Зо, якщо вони працюють в наукових установах бюджетної сфери:

$$Зо = \frac{М}{Тр} \cdot t, \quad (4.5)$$

де М – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.

У 2019 році величини окладів (разом з встановленими доплатами і надбавками) рекомендується брати в межах (5000…10000) грн. за місяць; Тр – число робочих днів в місяці; приблизно Тр = (21…23) дні; t – число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 4.6.

Таблиця 4.6 – Заробітна плата розробників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	17000	772	10	7727
Інженер-програміст	13000	590	10	5900
Всього:				13627

Основна заробітна плата робітників Зр, якщо вони беруть участь у виконанні даного етапу роботи і виконують роботи за робочими професіями у випадку, коли вони працюють в наукових установах бюджетної сфери, розраховується за формулою:

$$Зр = \sum_{i=1}^{n} ti \cdot Ci, \quad (4.6)$$

де ti – норма часу (трудомісткість) на виконання конкретної роботи, годин; n – число робіт по видах та розрядах; Сі – погодинна тарифна ставка робітника відповідного розряду, який виконує дану роботу. Сі визначається за формулою:

$$Ci = \frac{Мм \cdot Ki}{Тр \cdot Тзм}, \quad (4.7)$$

де Мм – розмір мінімальної заробітної плати за місяць, грн.; в 2019 році мінімальна заробітна плата становить – 4173 грн., Кі – тарифний коефіцієнт робітника відповідного розряду, Тр – число робочих днів в місяці; приблизно Тр = 21…23 дні; Тзм – тривалість зміни, зазвичай Тзм = 8 годин.

Таблиця 4.7 – Заробітна плата робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Налагоджувальні	8	5	34,4	1,45	275,2
Складальні	2	3	30,1	1,27	60,2
Всього					335,4

Додаткова заробітна плата Зд всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$Зд = 0{,}1 \cdot (Зр + Зо) = 0{,}1 \cdot (13627 + 335{,}4) = 1396{,}24 \text{ грн.} \quad (4.8)$$

Нарахування на заробітну плату Нзп розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

де Зо – основна заробітна плата розробників, грн.; Зр – основна заробітна плата робітників, грн.; Зд – додаткова заробітна плата всіх розробників та робітників, грн.; β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

$$Нзп = 0{,}22 \cdot (Зр + Зо + Зд) = 0{,}22 \cdot (13627 + 335{,}4 + 1624{,}5) = \\ = 3379 \text{ грн.} \quad (4.9)$$

Амортизація обладнання, комп'ютерів та приміщень А, які використовувались під час (чи для) виконання даного етапу роботи.

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$А = \frac{Ц \cdot На}{100} \cdot \frac{Т}{12},$$

де Ц – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.; На – річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що На = (10...25)%; Т – термін, використання обладнання, приміщень тощо, місяці.

Таблиця 4.8 - Амортизаційні відрахування

Найменування	Ціна, грн.	Норма амортизації, %	Термін використання, м.	Сума амортизації
ПК	11500	20	2	385
Інше обладнання	2700	10	2	45
Всього	430			

Витрати на силову електроенергію Ве, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$Ве = В \cdot П \cdot Ф \cdot Кп, грн$$

В – вартість 1 кВт-год. електроенергії, в 2019 р. В ≈ 8,45 грн./кВт; П – установлена потужність обладнання, кВт; Ф – фактична кількість годин роботи обладнання, годин, Кп – коефіцієнт використання потужності; Кп < 1.

Потужність обладнання складає – 0,7 кВт.

Кількість годин роботи складає – 320 годин.

Коефіцієнт викор. потужності – 0,9.

Ве=1703 грн.

Інші витрати Він охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо.

Інші витрати Ів можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$Ів = 1 \cdot (Зо + Зр) = 1 \cdot (13627 + 335{,}4) = 13962{,}4 \text{ грн.} \qquad (4.10)$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини (розділу, етапу) роботи – В.

$$В = 27924 \text{ грн.}$$

4.2.2 Розрахунок собівартості розробки моделі і засобу виявлення вразливостей у веб-додатках

Витрати на силову електроенергію Ве, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$Ве = В \cdot П \cdot Ф \cdot Кп, \text{грн}$$

В – вартість 1 кВт-год. електроенергії, в 2019 р. В ≈ 8,45 грн./кВт; П – установлена потужність обладнання, кВт; Ф – фактична кількість годин роботи обладнання, годин, Кп – коефіцієнт використання потужності; Кп < 1.

Потужність обладнання складає – 0,7 кВт.

Кількість годин роботи складає – 320 годин.

Коефіцієнт викор. потужності -0,9.

Ве=1703 грн.

Основна заробітна плата робітників Зр, якщо вони беруть участь у виконанні даного етапу роботи і виконують роботи за робочими професіями у випадку, коли вони працюють в наукових установах бюджетної сфери, розраховується за формулою:

$$Зр = \sum_{i=1}^{n} ti \cdot Ci, \qquad (4.11)$$

де ti – норма часу (трудомісткість) на виконання конкретної роботи, годин; n – число робіт по видах та розрядах; Сі – погодинна тарифна ставка робітника відповідного розряду, який виконує дану роботу. Сі визначається за формулою:

$$Сі = \frac{Мм \cdot Кі}{Тр \cdot Тзм}, \quad (4.12)$$

де Мм – розмір мінімальної заробітної плати за місяць, грн.; в 2019 році мінімальна заробітна плата становить – 4173 грн., Кі – тарифний коефіцієнт робітника відповідного розряду, Тр – число робочих днів в місяці; приблизно Тр = 21…23 дні; Тзм – тривалість зміни, зазвичай Тзм = 8 годин.

Таблиця 4.9 – Заробітна плата робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Налагоджувальні	8	5	34,4	1,45	275,2
Складальні	2	3	30,1	1,27	60,2
Всього					335,4

Додаткова заробітна плата Зд всіх робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$Зд = 0{,}1 \cdot (Зо) = 0{,}1 \cdot (335{,}4) = 33{,}54 \text{ грн.} \quad (4.13)$$

Нарахування на заробітну плату Нзп розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

де Зо – основна заробітна плата розробників, грн.; Зр – основна заробітна плата робітників, грн.; Зд – додаткова заробітна плата всіх розробників та робітників, грн.; β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

$$Нзп = 0{,}22 \cdot (Зо + Зд) = 0{,}22 \cdot (335{,}4 + 33{,}54) =$$

$$= 81 \text{ грн.} \quad (4.14)$$

«Загальновиробничі витрати» належать витрати: пов'язані з управлінням виробництвом (утримання працівників апарату управління виробництвом, оплата службових відряджень персоналу цехів, витрати на інформаційне забезпечення управління тощо); на повне відновлення та капітальний ремонт основних фондів загальновиробничого призначення; витрати некапітального характеру, пов'язані з удосконаленням технологій та організацією виробництва, поліпшенням якості продукції; на утримання, обслуговування, поточний ремонт виробничих приміщень; на контроль за виробничими процесами та кістю продукції.

Крім того, загальновиробничі витрати з розрахунку на одиницю продукції можна розрахувати за нормативами відносно до основної заробітної плати основних робітників, які виготовляють продукцію:

$$ЗВВ = Нв \cdot Зо, \quad (4.15)$$

Норматив загальновиробничих витрат для програмних продуктів становить 230-270%.

$$ЗВВ = 2{,}4 \cdot 335{,}4 = 805 \text{ грн},$$

Сума попередніх витрат утворює виробничу собівартість розробки:

$$Sв = 2957 \text{ грн.}$$

3. Розрахунок мінімальної ціни та чистого прибутку від реалізації розробки моделі і засобу виявлення вразливостей у веб-додатках.

Ціна – це грошовий вираз вартості товару (продукції, послуги). Вона завжди коливається навколо ціни виробництва (перетвореної форми вартості одиниці товару, що дорівнює сумі витрат виробництва й середнього прибутку) та відображає рівень суспільне необхідних витрат праці.

Виходячи з того, що розробки, як правило, приймаються та впроваджуються за завданням замовника, або коли результатом розробки є продукція, що підлягає державному регулюванню, то нижню межу ціни реалізації розробки можна розрахувати за формулою:

$$Ц = S_В \cdot \left(1 + \frac{Р}{100}\right) \cdot \left(1 + \frac{\omega}{100}\right), \quad (4.16)$$

де Sв – виробнича собівартість інноваційного рішення, грн.; Р – норматив рентабельності узгоджений із замовником або встановлений державою, (Р=30…60%); w – ставка податку на додану вартість, % (в 2019 році w=20%).

$$Ц = 2957 \cdot \left(1 + \frac{60}{100}\right) \cdot \left(1 + \frac{20}{100}\right) = 5677 \text{ грн.}$$

Чистий прибуток від реалізації розробки можна розрахувати за формулою:

$$П = \left(Ц - \frac{(Ц - МР) \cdot f}{100} - S_В - \frac{q \cdot S_В}{100}\right) \cdot \left(1 - \frac{h}{100}\right) \cdot РП, \quad (4.17)$$

де Ц – ціна розробки, грн.; МР – вартість матеріальних та інших ресурсів, що були придбані виробником для виготовлення розробки (МР=(0,1…0,2) Цр), грн.; f – зустрічна ставка податку на додану вартість, %; Sв – виробнича собівартість розробки, грн.; q – норматив, який визначає величину адміністративних витрат, витрат на збут та інші операційні витрати, % (рекомендовано q=5…10%); h – ставка податку на прибуток, %, РП – прогнозований попит продажів:

$$П = 20756 \text{ грн.}$$

4. Розрахунок терміну окупності коштів, вкладених в наукову розробку моделі і засобу виявлення вразливостей у веб-додатках

Термін окупності вкладених у реалізацію наукового проекту інвестицій Ток можна розрахувати за формулою:

$$Ток = \frac{В}{П} = \frac{27924}{20756} = 1{,}34 \text{ роки.} \quad (4.18)$$

Оскільки Ток < 3 років, то фінансування даної наукової розробки моделі і засобу виявлення вразливостей у веб-додатках є доцільним.

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи проведено аналіз літературних джерел та розглянуто ключові поняття. Розглянуті найпопулярніші загрози веб-ресурсів. Досліджено те, яким чином вони впливають на самі веб-ресурси, а також користувачів, які використовують вразливі сайти. Також проаналізовані існуючі методи виявлення цих загроз. Була поставлена проблема та запропонований шлях її вирішення.

Відповідно особливостям кожної з загроз було створено моделі і алгоритми їх визначення. Для визначення кріптоджекера використовувалась модель, яка враховує зростання навантаження і приймає рішення згідно з піковими показниками. Для перевірки на наявність XSS-атак було обрано рішення, яке вбудовує ін'єкцію у перевіряємий сайт. В залежності від отриманих у якості відповіді даних, приймається рішення. Для виявлення SQL-ін'єкцій використовувався алгоритм порівняння, який зіставляє множини елементів DOM-дерев, які були отримані після запитів з ін'єкцією та звичайних. На основі різниці цих множин робиться висновок.

Були обрані середовище та мова програмування. Відповідно до завдання магістерської кваліфікаційної дипломної роботи, створені моделі для виявлення кожної загрози і згідно цих моделей розроблено програмний засіб виявлення вразливостей у Web-додатках для покращення стану кібербезпеки. Даний програмний засіб протестовано та продемонстровано послідовність роботи у тестовій частині.

Даний програмний засіб може використовуватись тестувальниками або розробниками сайтів на cms, які часто мають вразливість до цих атак, а також для звичайного сканування на наявність вразливостей перед використанням обраного ресурсу, що може упередити втрату, доступність чи витік особистої інформації та не нанести шкоду апаратній частині комп'ютера чи ноутбука.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1) Все про кріптоджекінг [Електронний ресурс]. –Режим доступу: URL: https://ru.malwarebytes.com/cryptojacking/

2) Феномен кріптовалют: досвід системного опису [Електронний ресурс]. – Режим доступу: URL: https://cyberleninka.ru/article/n/fenomen-kriptovalyut-opyt-sistemnogo-opisaniya

3) Кріптоджекінг на підйомі [Електронний ресурс]. – Режим доступу: URL: https://www.kaspersky.ru/blog/cryptojacking-rsa2019/22390/

4) NoCoin documentation [Електронний ресурс]. – Режим доступу: URL: https://github.com/keraf/NoCoin/

5) ASIC-Майнінг [Електронний ресурс]. – Режим доступу: URL: https://cryptonet.biz/ru/asic-majning-kak-rabotaet-asic-majner-i-naskolko-vygoden-segodnya/

6) Як майнінг впливає на ресурс відеокарт [Електронний ресурс]. – Режим доступу: URL: https://hype.ru/@boevoy-homyak/kak-mayning-vliyaet-na-resurs-videokart-y1qys0r8

7) Як боротись із кріптоджекінгом [Електронний ресурс]. – Режим доступу: URL: https://medium.com/crypto.ff

8) Як захистити веб-сайт від прихованого майнінгу [Електронний ресурс]. – Режим доступу: URL: https://www.anti-malware.ru/practice/methods/How-to-protect-your-website-from-cryptojacking-attacks

9) Нелігетимний майнінг [Електронний ресурс]. – Режим доступу: URL: https://itpro.ua/post/spetsialisty_po_predotvrashcheniyu_kiberatak_fiksiruyut_uvelichenie_ugrozy_nelegitimnogo_maininga_kriptodzhekinga_v_setyakh_kommercheskikh_i_gosudarstvennykh_organizatsii/

10) Які найнебезпечніші вразливості сайтів існують? [Електронний ресурс]. – Режим доступу: URL:https://hyperhost.ua/info/kakie-samyie-opasnyie-uyazvimosti-saytov-sushhestvuyut/

11) 9 з 10 сайтів мають вразливості [Електронний ресурс]. – Режим доступу: URL: https://xakep.ru/2008/03/26/42963/

12) Тімохович О. С., Соколін Д.Д. Методи комплексного забезпечення інформації SQL-сервера від атак типу SQL-ін'єкцій //Інститут інформатики і комунікацій, 2016.

13) Атаки на сайты [Електронний ресурс]. – Режим доступу: URL: https://www.anti-malware.ru/threats/websites-attacks

14) Як працює міжсайтовий скриптинг [Електронний ресурс]. – Режим доступу: URL: https://wiki.rookee.ru/cross-site-scripting/

15) Тепляков С.П., Тімохович О. С. Аналіз і методи захисту веб-додатків від атак типу XSS //Державний інститут ім. академіка М.Ф.Решетньова, 2016

16) Voitovych O. P. SQL injection prevention system / O. P. Voitovych, O. S. Yuvkovetskyi, L. M. Kupershtein // 2016 International Conference Radio Electronics & Info Communications (UkrMiCo), 2016 - P: 1-4, DOI: 10.1109/UkrMiCo.2016.7739642

17) Voitovych O. P. SQL injection prevetion system / Voitovych O. P., Yuvkovetskyi O.S. // Тези доповідей П'ятої Міжнародної науково-практичної конференції «Методи та засоби кодування, захисту й ущільнення інформації» м. Вінниця, 19-21 квітня 2016 року. - Вінниця: ВНТУ, 2016. - С. 82-84.

18) Що таке Selenium WebDriver? [Електронний ресурс]. – Режим доступу: URL:https://kreisfahrer.gitbooks.io/seleniumwebdriver/content/webdriver_intro/webdriver_obzor_i_printsip_raboti.html

19) Носіров З. А., Ажмухамедов І. М. Виявлення XSS-вразливостей на основі аналізу повної карти веб-додатка // Системи управління, зв'язку та безпеки. 2018.

ДОДАТКИ

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра захисту інформації

ЗАТВЕРДЖУЮ

Завідувач кафедри ЗІ

д. т. н., проф.

_____ В. А. Лужецький

"____" _____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему

"Модель і засіб виявлення вразливостей у Web-додатках "

08-20.МКР.016.00.000.ПЗ

Розробив студент групи 1БС-18м

_____ Юсуфов Р.Ю.

Керівник магістерської кваліфікаційної роботи д. т. н., проф.

_____ Войтович О. П.

_____ 2019 р.

Вінниця 2019

1 Найменування та область застосування

Модель і засіб виявлення вразливостей у Web-додатках

2 Підстави для розробки

Розробка виконується на основі наказу ректора №254 від 02.10.2019 року.

3 Мета та призначення

3.1 Мета дослідження

Покращення безпеки у групах та спільнотах соціальних мереж за допомогою пошуку та аналізу центрів інформаційного впливу

3.2 Призначення

Знаходження осіб, які мають найбільший вплив у групах та спільнотах соціальних мереж та можуть бути потенційно засланими «агентами»

4 Джерела розробки

4.1 Все про кріптоджекінг [Електронний ресурс]. –Режим доступу: URL: https://ru.malwarebytes.com/cryptojacking/

4.2 Тімохович О. С., Соколін Д.Д. Методи комплексного забезпечення інформації SQL-сервера від атак типу SQL-ін'єкцій //Інститут інформатики і комунікацій, 2016.

4.3 Як захистити веб-сайт від прихованого майнінгу [Електронний ресурс]. – Режим доступу: URL: https://www.anti-malware.ru/practice/methods/How-to-protect-your-website-from-cryptojacking-attacks

4.4 Як захистити веб-сайт від прихованого майнінгу [Електронний ресурс]. – Режим доступу: URL: https://www.anti-malware.ru/practice/methods/How-to-protect-your-website-from-cryptojacking-attacks

4.5 Носіров З. А., Ажмухамедов І. М. Виявлення XSS-вразливостей на основі аналізу повної карти веб-додатка // Системи управління, зв'язку та безпеки. 2018.

5 Технічні вимоги

5.1 Загальні вимоги

5.1.1 Даний програмний засіб повинен реалізовувати визначення вразливостей до SQL-ін'єкцій, XSS-атак і визначення наявності кріптоджекера на веб-ресурсах.

5.1.2 Реалізація не повинна вимагати спеціальних ліцензійних програмних додатків.

5.1.3 Реалізація повинна забезпечити візуальне відображення у вигляді графів.

5.2 Вимоги до тестування

5.2.1 Тестування повинно складатись з отримання даних про сканування кожної з загроз.

5.2.2 Провести тестування із заздалегіть вразливими сайтами і невразливими для порівняння результату спрацювання.

5.2.3 Створення вибірки сканування і визначення похибки для кожної моделі.

5.3 Вимоги до програмної реалізації

5.3.1 Використання ОС Windows.

5.3.2 Визначення вразливостей веб-ресурсів на загрози SQL-ін'єкцій, XSS-атак і наявність кріптоджекінгу.

6 Стадії та етапи розробки

№	Зміст	Початок	Закінчення	Результат
1	Огляд літературних джерел, аналіз і формулювання вимог до тестування, розробка ТЗ.	09.03	18.03	Розділ ПЗ „Аналіз літературних джерел"
2	Аналіз понять «соціальна мережа», «інформаційне протиборство» та їх взаємозв'язок	20.03	27.03	Розгляд основних понять, постановка завдання. Розділ ПЗ.
3	Теоретична частина. Алгоритми роботи, методи та математичні моделі	28.03	24.04	Використання математичних моделей та методів

				для реалізації пошуку «агентів»
4	Тестування роботи програми. Формулювання висновків відповідно проведеної роботи. Оформлення ПЗ	25.04	15.06	Пояснювальна записка, програмний засіб, інструкції до програми

7 Порядок контролю та приймання

7.1 До приймання дипломної роботи представляється:

- ПЗ до дипломної роботи;
- робоча реалізація;
- результати тестування;
- ілюстративні матеріали для захисту.

7.2 Рубіжний контроль керівника _____

7.3 Попередній захист на кафедрі _____

7.4 Захист на ДЕК _____

ДОДАТОК Б

ЛІСТИНГ ПРОГРАМИ

```
const { writeFile } = require("fs");
const { promisify } = require("util");

const cors = require("cors");
const osu = require("node-os-utils");
const express = require("express");
const bodyParser = require("body-parser");
const { Builder, By, Key, until } = require("selenium-webdriver");
const { xssDiff, sqlDiff } = require("./diffBetweenSnapshots");
let app = express();

app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

let counter = 0;

let writeAsync = promisify(writeFile);

const asyncSleep = time =>
  new Promise(resolve => setTimeout(_ => resolve(), time));

// POST param: url - string
app.post("/detect_vulnerability", function(req, res) {
  let urlForDetect = req.body.url;
  (async function example() {
    let driver = await new Builder().forBrowser("chrome").build();
    try {
      await driver.get("http://www.google.com/ncr");
      await driver.findElement(By.name("q")).sendKeys("webdriver", Key.RETURN);
      let networkData = await driver.executeScript(
        `var performance = window.performance || window.mozPerformance || window.msPerformance || window.we
bkitPerformance || {}; var network = performance.getEntries() || {}; return network;`
      );

      await driver.wait(until.titleIs("webdriver - Google Search"), 5000);
      await driver.findElement(By.name("q")).clear();
      await driver.findElement(By.name("q")).sendKeys("Ruslik liubit Tarasa");


      let allInputs = await driver.findElements(By.css("input"));
    } finally {
    }
  })();
  const xssAlert = `<script>alert(33)</script>`;
  const xssImage = `<img src="http://fsd.xo/ewefe" onerror="javascript:alert(33)"/>`;
});

app.post("/test_endpoint", async function(req, res) {
  let freeAmountCPU = [];
  let urlForDetect = req.body.url;
await (async function example() {
  let cpu = await osu.cpu.free();
  freeAmountCPU.push(cpu);
  let driver = await new Builder().forBrowser("chrome").build();
  const xssAlert = `<script>alert(33);window.ef=42</script>`;
  const xssImage = `<img src="http://fsd.xo/ewefe" onerror="javascript:alert(33)"/>`;
  try {
    await driver.get(urlForDetect);

    let el = await driver.findElement(By.css("body"));
    await driver.wait(until.elementIsVisible(el), 3000);
    let allInputs = await driver.findElements(By.css("input"));

    let submitInputs = await allInputs.filter(async i => {
      return (await i.getAttribute("type")) == "submit";
    });
    let textInputs = await allInputs.filter(async i => {
      return (await i.getAttribute("type")) == "text";
    });
```

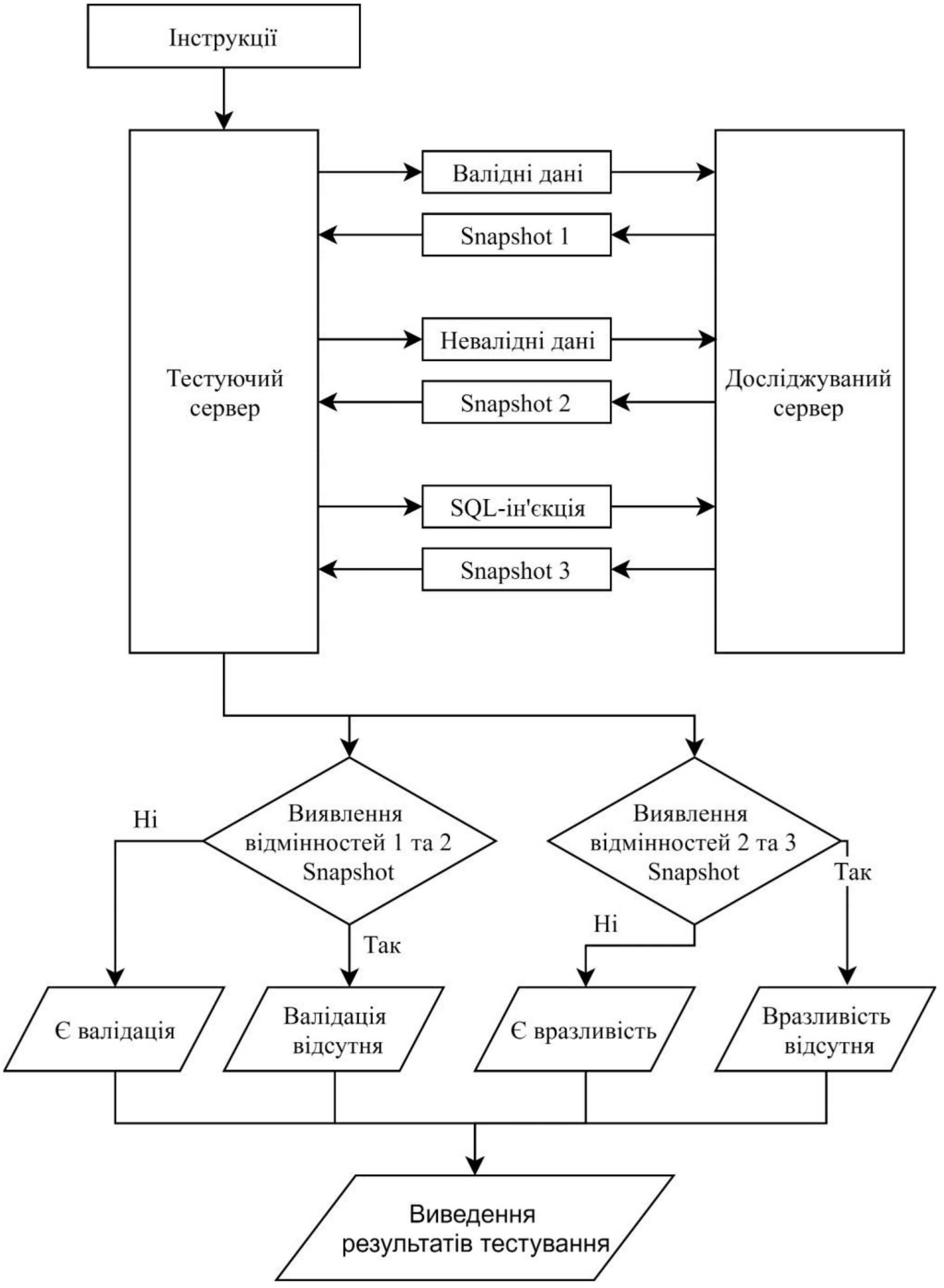
```
    for (let input of textInputs) {
      input.sendKeys(xssAlert);
    }
    await textInputs[0].sendKeys(Key.RETURN);
    let timeouts = await driver.manage().getTimeouts();

    let source = await driver.getPageSource();
    await writeAsync(`snapshots/${new Date().toISOString()}`, source);

  } catch (err) {
    res.send({ results: ["catch block"] });
  } finally {
    res.send({ results: ["finally block"] });
  }
})();
res.send({ results: freeAmountCPU });
});

app.post("/detect_miner", async function(req, res) {
  let freeAmountCPU = [];
  let driver = await new Builder().forBrowser("chrome").build();
  try {
    let urlForDetect = req.body.url;
    let cpu = await osu.cpu.free();
    freeAmountCPU.push(cpu);
    await driver.get(urlForDetect);
    let el = await driver.findElement(By.css("body"));
    await driver.wait(until.elementIsVisible(el), 3000);
    cpu = await osu.cpu.free();
    freeAmountCPU.push(cpu);
    await asyncSleep(5000);
    cpu = await osu.cpu.free();
    freeAmountCPU.push(cpu);
    await asyncSleep(5000);
    cpu = await osu.cpu.free();
    freeAmountCPU.push(cpu);
    await asyncSleep(5000);
    cpu = await osu.cpu.free();
    freeAmountCPU.push(cpu);
    // res.send({ results: freeAmountCPU });
  } catch (err) {
    res.send({ results: ["catch block"] });
  } finally {
    // res.send({ results: ["finally block"] });

    await driver.quit();
    await asyncSleep(5000);
    cpu = await osu.cpu.free();
    freeAmountCPU.push(cpu);
    res.send({ results: freeAmountCPU });
  }
});

app.post("/check_diff", async function(req, res) {
  let driver = await new Builder().forBrowser("chrome").build();
  let [first, second] = req.body.url;
  try {
    await driver.get(first);
    let firstResult = await driver.getPageSource();
    firstResult = firstResult.slice(firstResult.indexOf("<body"));
    console.log("firstres:", typeof firstResult);

    await driver.get(second);
    let secondResult = await driver.getPageSource();
    secondResult = secondResult.slice(secondResult.indexOf("<body"));

    let difference = xssDiff(firstResult, secondResult);
    console.log(difference);
    res.send({ result: difference, snapshot: firstResult });
  } catch (error) {
  } finally {
    driver.quit();
  }
});

app.post("/for_xss", function(req, res) {
  counter++;
  console.log(`POST /for_xss called, counter: ${counter}`);
  console.log(
```

```
    // `Data from vulnarable webapp: ${JSON.stringify(req.body)}`
    console.log(
      `Data from vulnarable webapp: `,
      req.body
      // .cookies,
      // req.body.localStorage
    )
  );
  res.send({ ok: "vulnerable" });
});

app.post("/check_xss", async function(req, res) {
  console.log(`POST /check_xss called, counter: ${counter}`);
  let { url, query } = req.body;
  let driver = await new Builder().forBrowser("chrome").build();
  const xssFetch = `<script>fetch("http://localhost:4231/for_xss",{method:"POST","Content-
Type":"application/json",body:JSON.stringify({"cookies":document.cookie,"localStorage":JSON.stringify(local
Storage)})}).then((res)=>{console.log(res)})</script>`;
  const xssFetchAwait = `<script>await fetch("http://localhost:4231/for_xss")</script>`;
  try {
    // check Query Params for XSS vulnerability
    if (query) {
      for (let q of query) {
        let prevValue = counter;
        let injectedURL = `${url}?${q}=${xssFetch}`;
        await driver.get(injectedURL);
        await asyncSleep(5000);
        if (counter > prevValue) {
          res.send({ url: injectedURL, vulnarable: true });
          return;
        }
      }
    }

    // check Forms for XSS vulnerability
    await driver.get(`${url}`);
    let el = await driver.findElement(By.css("body"));
    await driver.wait(until.elementIsVisible(el), 3000);
    let allInputs = await driver.findElements(By.css("input"));
    let textInputs = await allInputs.filter(async i => {
      return (await i.getAttribute("type")) == "text";
    });
    for (let input of textInputs) {
      await input.sendKeys(xssFetch);
    }
    let currentURL = await driver.getCurrentUrl();
    console.log("currentURL:", currentURL);

    let prevCounter = counter;
    for (let input of textInputs) {
      if ((await driver.getCurrentUrl()) === url) {
        await input.sendKeys(Key.RETURN);
      } else {
        break;
      }
    }
    console.log("before asyncSleep(4000)");

    await asyncSleep(4000);
    // res.send({ s: 4 });
    console.log("after asyncSleep(4000)");

    if (counter > prevCounter) {
      res.send({ url: url, vulnarable: true });
    } else {
      res.send({ url: url, vulnarable: false });
    }
  } catch (error) {
    console.log("catch:", error);
  } finally {
    console.log("finally block");

    // driver.quit();
  }
});

app.post("/check_sql", async function(req, res) {
  let { url, query } = req.body;
  let driver = await new Builder().forBrowser("chrome").build();
```

```
const injection = [" or 1=1"]; // SQLtoInject
// check Query Params at first
if (url && url !== []) {
  // for (let injection of SQLtoInject) {
  let URLtoCheck = `${url}${injection[0]}`;
  await driver.get(URLtoCheck);
  console.log("await driver.get(URLtoCheck);");
  let injectionResult = await driver.getPageSource();
  injectionResult = injectionResult.slice(injectionResult.indexOf("<body"));
  console.log("injectionResult");
  // await asyncSleep(2000);
  await driver.get(url);
  console.log("await driver.get(url);");
  let defaultResult = await driver.getPageSource();
  defaultResult = defaultResult.slice(defaultResult.indexOf("<body"));
  console.log("defaultResult");
    let difference = injectionResult.length - defaultResult.length; // sqlDiff(defaultResult, injectionResu
lt);
    console.log(difference);

    res.send({
      difference: difference,
      injectedLength: injectionResult.length,
      withoutInjectionLength: defaultResult.length,
      vulnarable: difference > 0
    });
    return;
  }
});
app.listen(4231, function() {
  console.log("Started on port 4231");
});
```

ГРАФІЧНА ЧАСТИНА

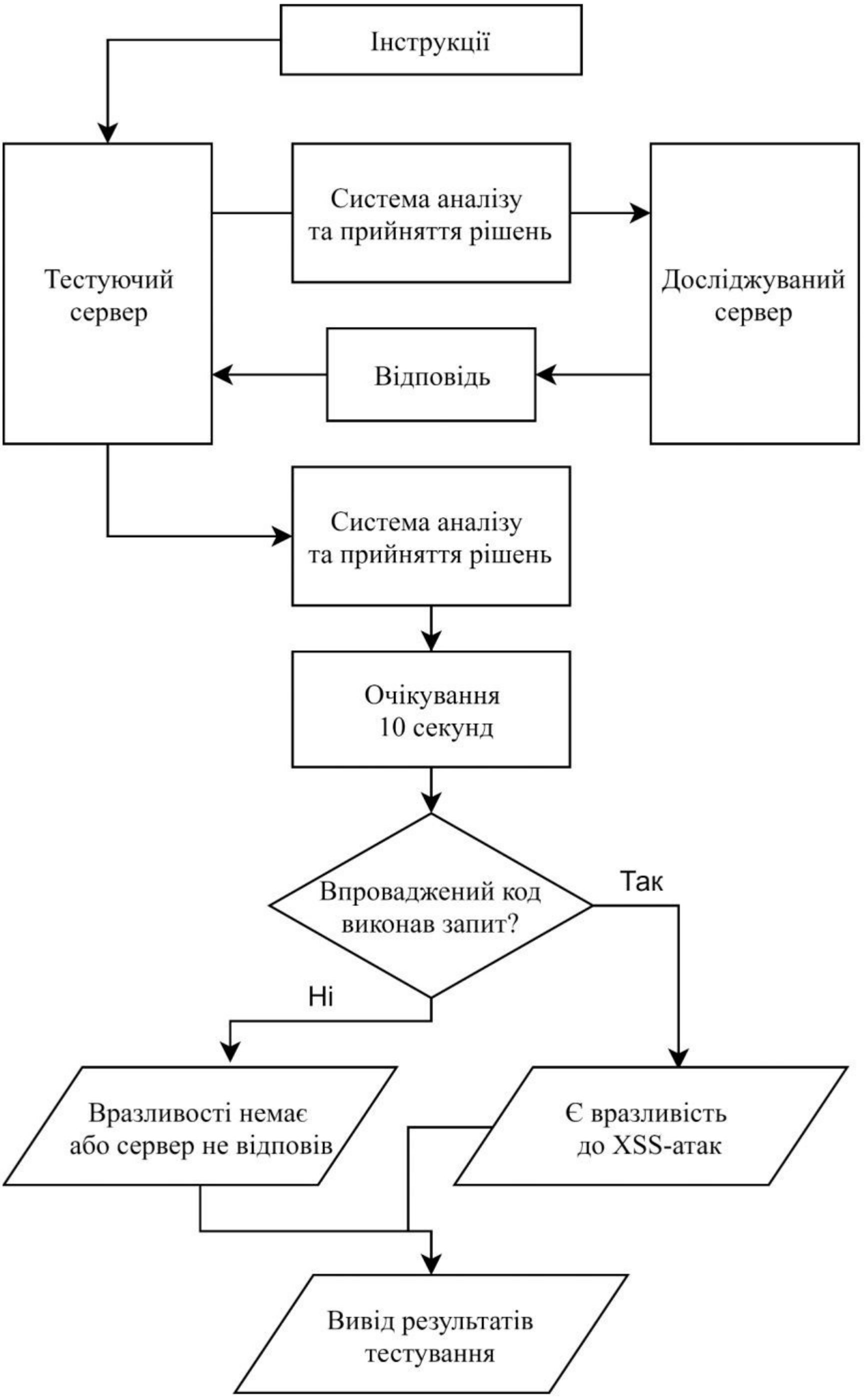
Модель виявлення вразливостей до SQL-ін’єкцій



					08-20.ТСЗСЗвК.018.16.116 ІЧ1
Змн	Арк.	№ докум.	Підпис	Дата	
Розроб.		Юсуфов Р.Ю.			Модель і засіб виявлення вразливостей у Web-додатках. Модель виявлення вразливостей до SQL-ін'єкцій
Перевір.		Войтович О.П.			
Рецензент					
Н. Контр.		Войтович О.П.			
Затверд.		ЛужецькийВ.А.			

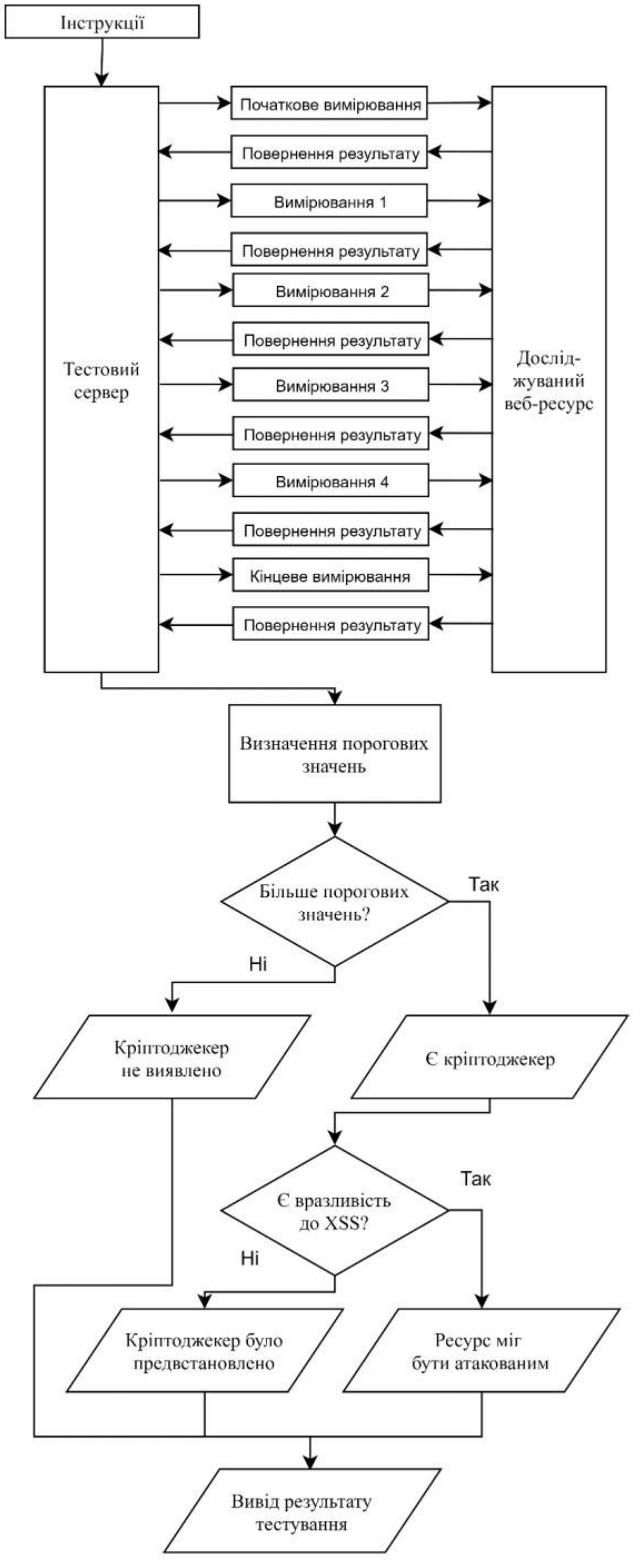
Літ.	Маса	Масштаб

Модель виявлення вразливостей до XSS-атак



					08-20.ТСЗСЗвК.018.16.116 ІЧ2		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Юсуфов Р.Ю.			Модель і засіб виявлення вразливостей у Web-додатках. Модель виявлення вразливостей до XSS-атак	Літ.	Маса / Масштаб
Перевір.		Войтович О.П.					
Н. Контр.		Войтович О.П.					
Затверд.		ЛужецькийВ.А.					

Модель виявлення кріптоджекера

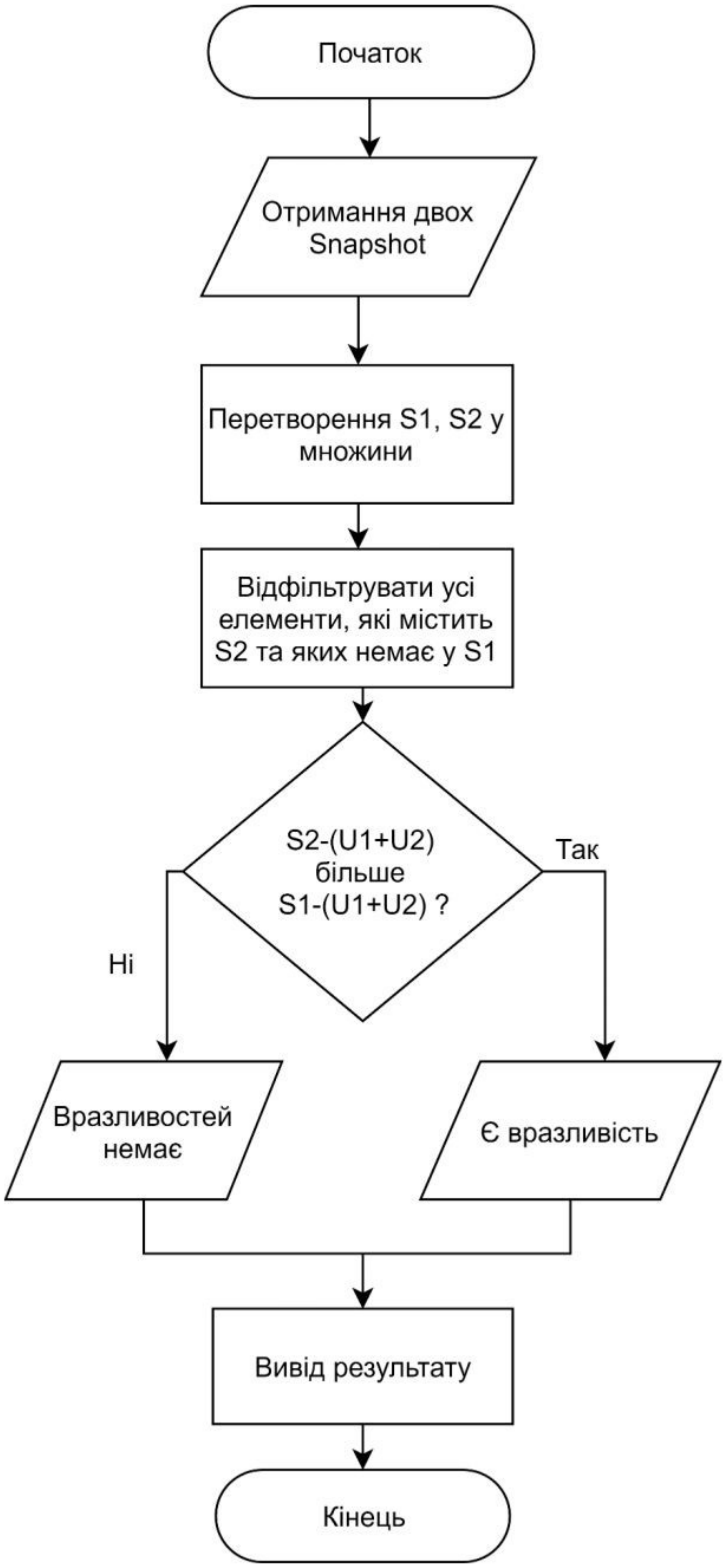


					08-20.ТСЗСЗвК.018.16.116 ІЧЗ		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Юсуфов Р.Ю.			Модель і засіб виявлення вразливостей у Web-додатках. Модель виявлення кріптоджекера	Літ.	Маса / Масштаб
Перевір.		Войтович О.П.					
Рецензент							
Н. Контр.		Войтович О.П.					
Затверд.		ЛужецькийВ.А.					

					08-20.ТСЗСЗвК.018.16.116 ІЧЗ
Змн	Арк.	№ докум.	Підпис	Дата	
Розроб.		Юсуфов Р.Ю.			Модель і засіб виявлення вразливостей у Web-додатках. Модель виявлення кріптоджекера
Перевір.		Войтович О.П.			
Н. Контр.		Войтович О.П.			
Затверд.		ЛужецькийВ.А.			

Літ.	Маса	Масштаб

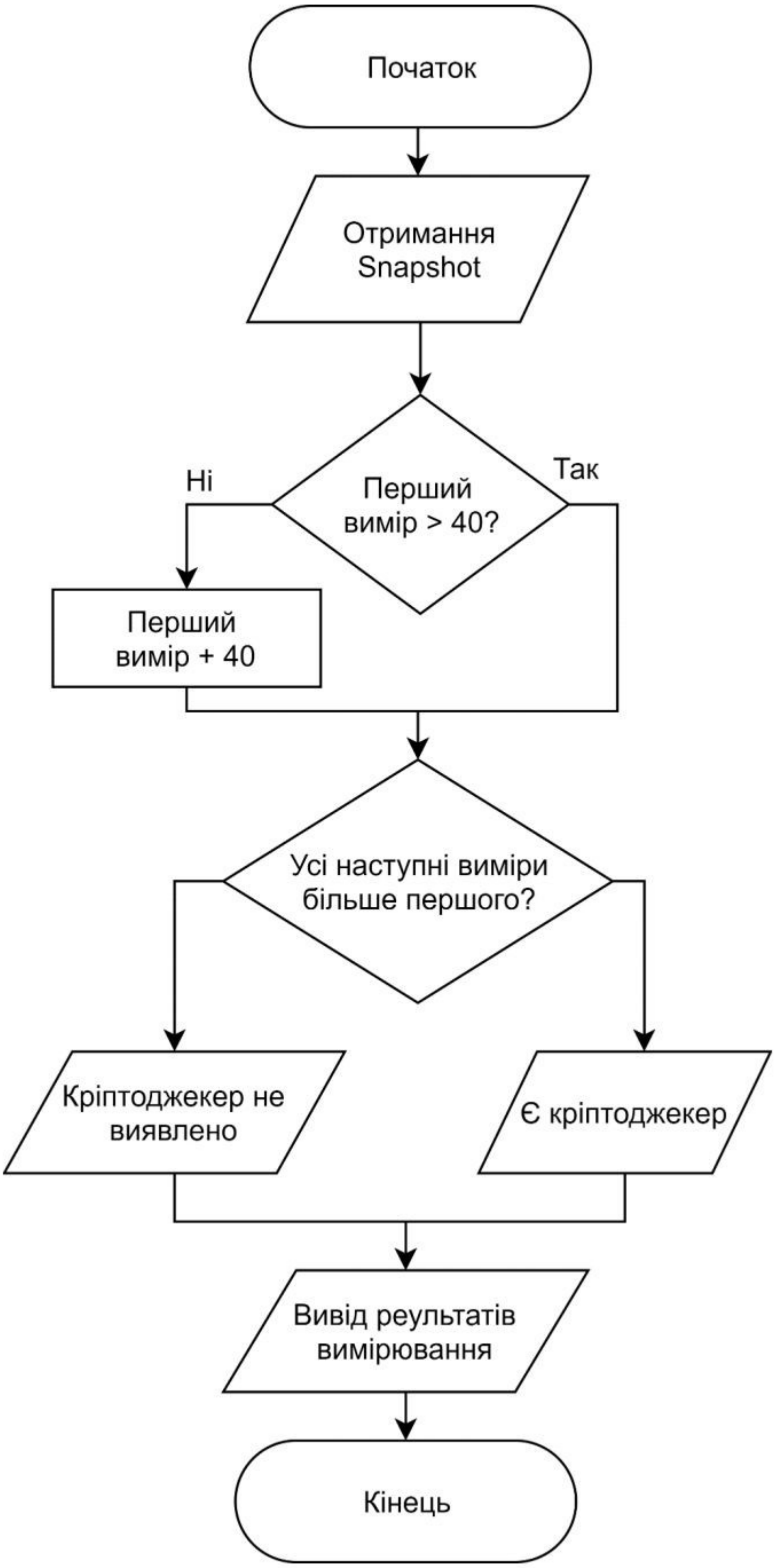
Алгоритм пошуку різниці множин



					08-20.ТСЗСЗвК.018.16.116 ІЧ4
Змн	Арк.	№ докум.	Підпис	Дата	
Розроб.		Юсуфов Р.Ю.			Модель і засіб виявлення вразливостей у Web-додатках. Алгоритм пошуку різниці множин
Перевір.		Войтович О.П.			
Рецензент					
Н. Контр.		Войтович О.П.			
Затверд.		ЛужецькийВ.А.			

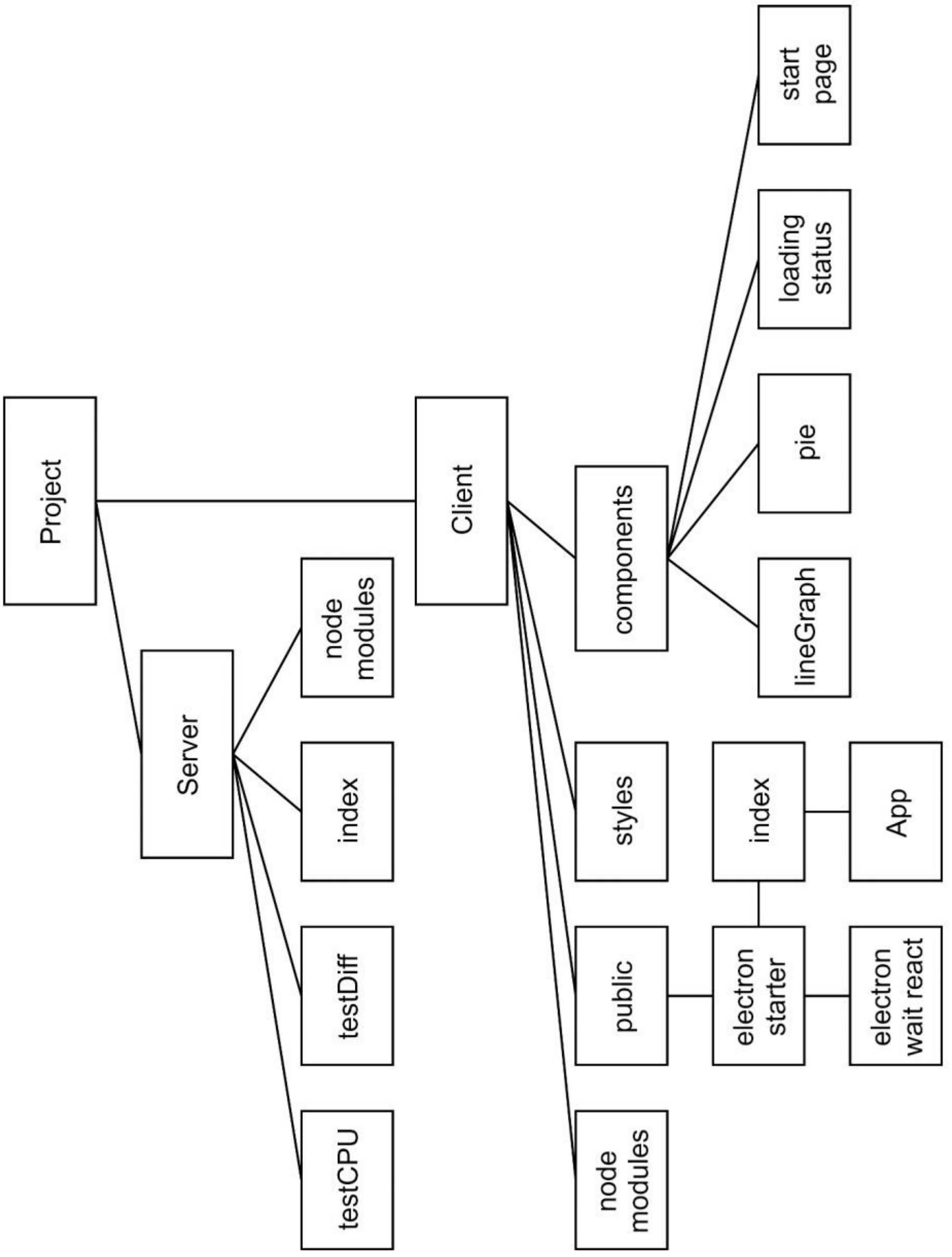
Літ.	Маса	Масштаб

Алгоритм порівняння значень із пороговим



					08-20.ТСЗСЗвК.018.16.116 ІЧ5		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Юсуфов Р.Ю.			Модель і засіб виявлення вразливостей у Web-додатках. Алгоритм порівняння значень із пороговим	Літ.	Маса / Масштаб
Перевір.		Войтович О.П.					
Рецензент							
Н. Контр.		Войтович О.П.					
Затверд.		ЛужецькийВ.А.					

Структура програмного засобу



					08-20.ТСЗСЗвК.018.16.116 ІЧ6		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Юсуфов Р.Ю.			Модель і засіб виявлення вразливостей у Web-додатках. Структура програмного засобу	Літ.	Маса / Масштаб
Перевір.		Войтович О.П.					
Рецензент							
Н. Контр.		Войтович О.П.					
Затверд.		ЛужецькийВ.А.					

Результати тестування програми

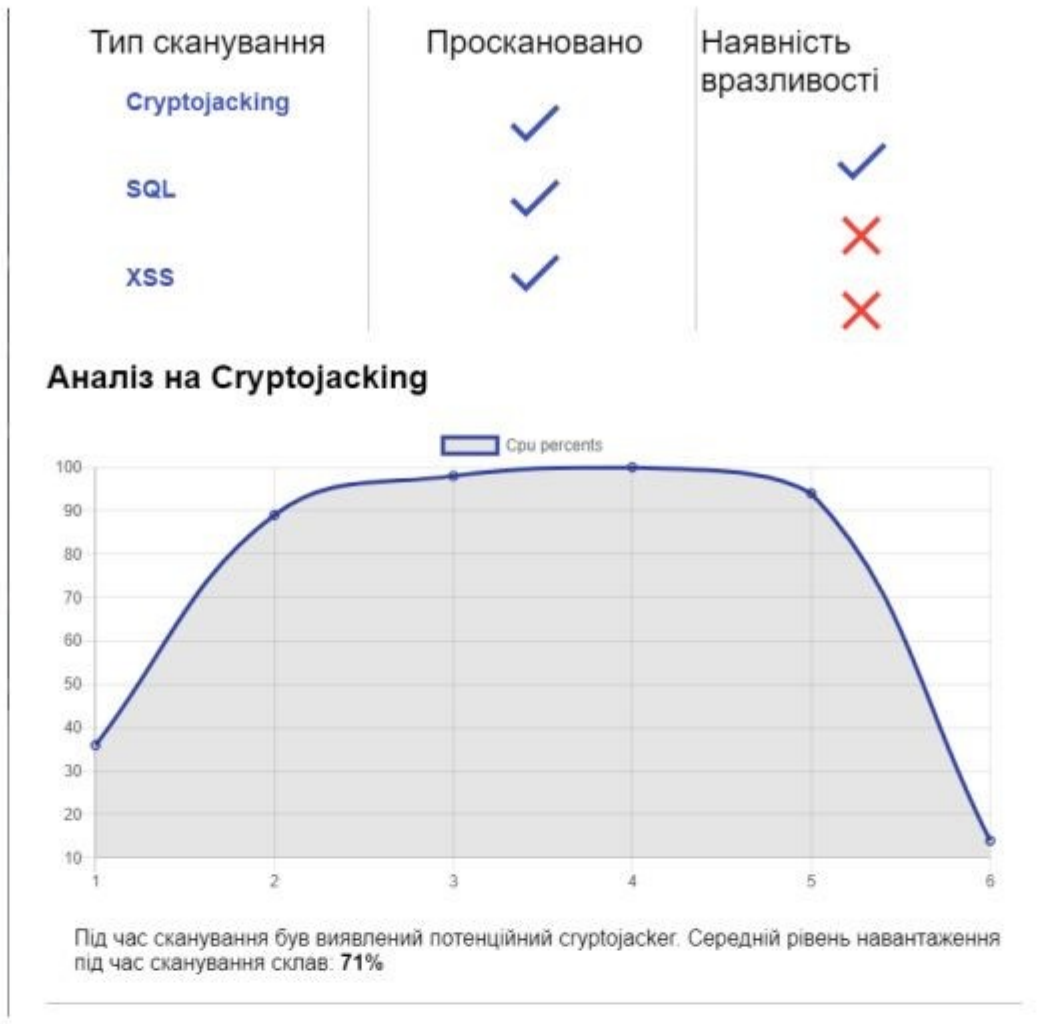


Рисунок 1 – Визначення кріптоджекера

Рисунок 2 – Визначення SQL-ін’єкції

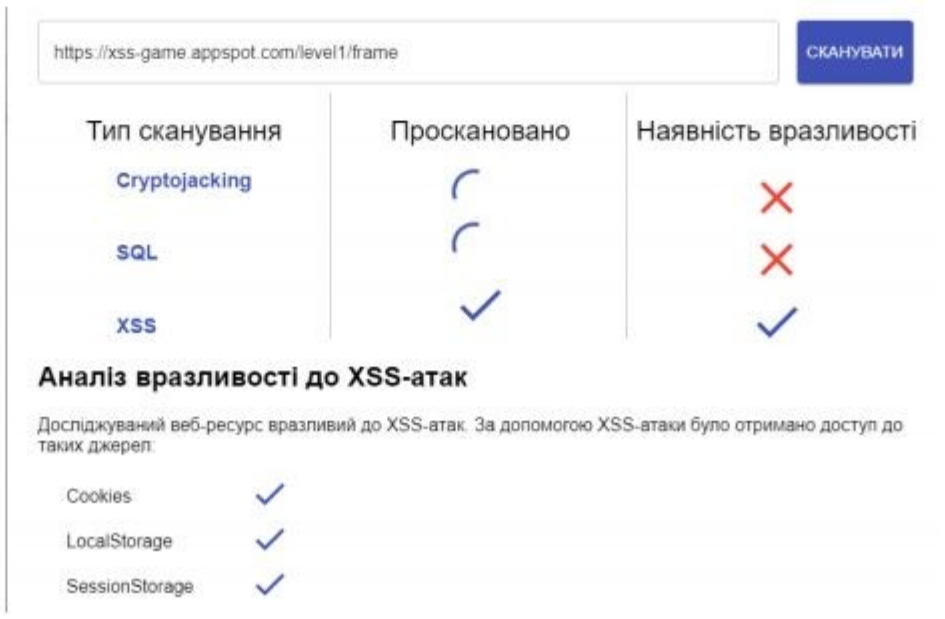


Рисунок 3 – Визначення вразливості до XSS-атаки

					08-20.ТСЗСЗвК.018.16.116 ІЧ7		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Юсуфов Р.Ю.			Модель і засіб виявлення вразливостей у Web-додатках. Результати тестування програми	Літ.	Маса / Масштаб
Перевір.		Войтович О.П.					
Рецензент							
Н. Контр.		Войтович О.П.					
Затверд.		ЛужецькийВ.А.					